REMARKS

Claims 13-17, 21-22, and 63-64 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over alleged AAPA (prior art in US 2005/0080951) in view of Vorbach (US 2007/0299993). This rejection is respectfully traversed.

Claim 13 recites "a first buffer having a first programmable buffer stage depth comprising a first number of buffer stages" and "a second buffer having a second programmable buffer stage depth comprising a second number of buffer stages." The Office Action admits that the AAPA fails to disclose these limitations. Office Action at 3. The Office Action relies on Vorbach's FIFO buffers as teaching these elements. According to the Office Action, Vorbach's FIFO buffers have adjustable latencies. The Office Action cites to paragraphs 114-117 of Vorbach as showing the missing features of claim 13 and to support its conclusion of obviousness.[1]

Applicant respectfully submits, however, that the Vorbach reference is not valid 102(e)/103(a) prior art against the present application. The Vorbach reference is a continuation-in-part of application no. 09/967,847, which was filed on September 28, 2001, and issued as U.S. Patent no. 7,210,129. The '847 application does not contain the disclosure that the Office Action now relies upon. The Vorbach reference also claims priority to provisional application no. 60/317,876, filed on September 7, 2001. Provisional application no. 60/317,876 has not been published and does not appear on the PAIR system. As such, there is no evidence that provisional application no. 60/317,876 contains the additional disclosure relied upon in the Office Action either.

The Vorbach reference also claims priority to PCT application PCT/EP02/2403, which was filed on March 5, 2002. PCT/EP02/2403 was published in German and cannot be a valid 102(e) reference. See 35 U.S.C. § 102(e)("an international application filed under the treaty defined in section 351(a) shall have the effects for the purposes of this subsection of an application filed in the United States only if the international application designated the United States and was

[1] Applicant notes that the Vorbach reference was filed without figures or a description of drawings section, yet the specification of the printed publication refers to many figures. Applicant believes that the Vorbach reference may not constitute enabling prior art.

published under Article 21(2) of such treaty in the English language."). Applicant provides a copy of PCT/EP02/2403 for the Examiner's convenience.[2] Accordingly, the effective date of the Vorbach reference is its 371 date of February 17, 2005, which is after the filing date of the present application. As such, the additional disclosure relied upon in the Office Action (i.e., paragraphs 114-117) was not prior in time to the subject matter of the present application and cannot be cited to cure the deficiencies of the AAPA.

Accordingly, claims 13-17, 21-22, and 63-64 are allowable over the cited combination. The rejection should be withdrawn and the claims allowed.

Claim 20 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over the AAPA in view of Vorbach and Jander (US 5,956,492). This rejection is respectfully traversed. Claim 20 depends from claim 13. As such, claim 20 is allowable over the AAPA and Vorbach combination. Jander does not cure the deficiencies of the AAPA and Vorbach combination. Accordingly, the rejection should be withdrawn and the claim allowed.

In view of the above, Applicant believes the pending application is in condition for allowance.

Dated: February 24, 2009

Respectfully submitted,

By _____

Gianni Minutoli
Registration No.: 41,198
DICKSTEIN SHAPIRO LLP
1825 Eye Street, NW
Washington, DC 20006-5403
(202) 420-2200
Attorney for Applicant

Attachment

[2] The English language Abstract portion of the PCT publication does not disclose programmable buffer stages or FIFO's with programmable latencies either.

ATTACHMENT

(12) NACH DEM VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES PATENTWESENS (PCT) VERÖFFENTLICHTE INTERNATIONALE ANMELDUNG

(19) Weltorganisation für geistiges Eigentum
Internationales Büro

(43) Internationales Veröffentlichungsdatum
12. September 2002 (12.09.2002)

PCT

(10) Internationale Veröffentlichungsnummer
WO 02/071249 A2

(51) Internationale Patentklassifikation[7]: G06F 15/80

(21) Internationales Aktenzeichen: PCT/EP02/02403

(22) Internationales Anmeldedatum:
5. März 2002 (05.03.2002)

(25) Einreichungssprache: Deutsch

(26) Veröffentlichungssprache: Deutsch

(30) Angaben zur Priorität:

101 10 530.4	5. März 2001 (05.03.2001)	DE
101 11 014.6	7. März 2001 (07.03.2001)	DE
PCT/EP01/06703	13. Juni 2001 (13.06.2001)	EP
101 29 237.6	20. Juni 2001 (20.06.2001)	DE
pct/EP01/115021.6	20. Juni 2001 (20.06.2001)	EP
101 35 210.7	24. Juli 2001 (24.07.2001)	DE
101 35 211.5	24. Juli 2001 (24.07.2001)	DE
EP0108534	24. Juli 2001 (24.07.2001)	EP
101 39 170.6	16. August 2001 (16.08.2001)	DE
101 42 231.8	29. August 2001 (29.08.2001)	DE
101 42 894.4	3. September 2001 (03.09.2001)	DE
101 42 903.7	3. September 2001 (03.09.2001)	DE
101 42 904.5	3. September 2001 (03.09.2001)	DE
60/317,876	7. September 2001 (07.09.2001)	US
101 44 732.9	11. September 2001 (11.09.2001)	DE
101 44 733.7	11. September 2001 (11.09.2001)	DE
101 45 792.8	17. September 2001 (17.09.2001)	DE
101 45 795.2	17. September 2001 (17.09.2001)	DE
101 46 132.1	19. September 2001 (19.09.2001)	DE
09/967,847	28. September 2001 (28.09.2001)	US
ep01/11299	30. September 2001 (30.09.2001)	EP
ep01/11593	8. Oktober 2001 (08.10.2001)	EP
101 54 259.3	5. November 2001 (05.11.2001)	DE
101 54 260.7	5. November 2001 (05.11.2001)	DE
ep01/129923.7	14. Dezember 2001 (14.12.2001)	EP
ep02/001331.4	18. Januar 2002 (18.01.2002)	EP
102 02 044.2	19. Januar 2002 (19.01.2002)	DE
102 02 175.9	20. Januar 2002 (20.01.2002)	DE
102 06 653.1	15. Februar 2002 (15.02.2002)	DE
102 06 856.9	18. Februar 2002 (18.02.2002)	DE
102 06 857.7	18. Februar 2002 (18.02.2002)	DE
102 07 225.6	21. Februar 2002 (21.02.2002)	DE
102 07 224.8	21. Februar 2002 (21.02.2002)	DE
102 07 226.4	21. Februar 2002 (21.02.2002)	DE
102 08 435.1	27. Februar 2002 (27.02.2002)	DE
102 08 434.3	27. Februar 2002 (27.02.2002)	DE

(71) Anmelder *(für alle Bestimmungsstaaten mit Ausnahme von US)*: **PACT INFORMATIONSTECHNOLOGIE GMBH** [DE/DE]; Muthmannstrasse 1, 80939 München (DE).

(72) Erfinder; und

(75) Erfinder/Anmelder *(nur für US)*: **VORBACH, Martin** [DE/DE]; Gotthardstrasse 117a, 80689 München (DE). **BAUMGARTE, Volker** [DE/DE]; Barharossaplatz 14, 81677 München (DE). **NÜCKEL, Armin, Dr.** [DE/DE]; Drosselweg 4, 76777 Neupotz (DE). **MAY, Frank** [DE/DE]; An der Tuchbleiche 12, 81927 München (DE).

(74) Anwalt: **PIETRUK, Claus, Peter**; Heinrich-Lilienfein-Weg 5, 76229 Karlsruhe (DE).

(81) Bestimmungsstaaten *(national)*: AE, AG, AL, AM, AT, AT (Gebrauchsmuster), AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DE (Gebrauchsmuster), DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO,

[Fortsetzung auf der nächsten Seite]

(54) Title: METHOD AND DEVICES FOR TREATING AND/OR PROCESSING DATA

(54) Bezeichnung: VERFAHREN UND VORRICHTUNGEN ZUR DATENBE- UND/ODER VERARBEITUNG

(57) Abstract: The invention relates to procedures and methods for administering and transferring data within multi-dimensional systems consisting of transmitters and receivers. The division of a data stream into several independent branches and the subsequent combination sequential collection of the individual branches to form a data stream can be carried out in a simple manner. The individual data streams are re-combined in a correct time sequence. Said inventive method is particularly useful for processing re-entrant codes and is suitable for configurable architectures wherein efficient control of the configuration and reconfiguration is highly important.

(57) Zusammenfassung: Die Erfindung beschriebt Verfahren und Methoden zur Verwaltung und zum Transfer von Daten innerhalb von mehrdimensionalen Anordnungen von Sendern und Empfängern. Das Aufteilen eines Datenstromes in mehrere unabhängige Zweige und das nachfolgende Zusammenfassen der einzelnen Zweige zu einem Datenstrom soll einfach durchführbar sein, wobei die einzelnen Datenströme in der korrekten zeitlichen Reihenfolge wieder zusammengefasst werden. Insbesondere zur Abarbeitung von reentrantem Code ist dieses Verfahren von besonderer Wichtigkeit. Das beschriebene Verfahren ist insbesondere für konfigurierbare Architekturen geeignet, die effiziente Steuerung der Konfiguration und Rekonfiguration findet besondere Beachtung.

WO 02/071249 A2

RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Bestimmungsstaaten *(regional)*: ARIPO-Patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), eurasisches Patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), europäisches Patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI-Patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Erklärung gemäß Regel 4.17:

— *hinsichtlich der Berechtigung des Anmelders, ein Patent zu beantragen und zu erhalten (Regel 4.17 Ziffer ii) für die folgenden Bestimmungsstaaten AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW, ARIPO-Patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), eurasisches Patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), europäisches Patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI-Patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG)*

Veröffentlicht:

— *ohne internationalen Recherchenbericht und erneut zu veröffentlichen nach Erhalt des Berichts*

Zur Erklärung der Zweibuchstaben-Codes und der anderen Abkürzungen wird auf die Erklärungen ("Guidance Notes on Codes and Abbreviations") am Anfang jeder regulären Ausgabe der PCT-Gazette verwiesen.

Verfahren und Vorrichtungen zur Datenbe- und/oder Verarbeitung

Beschreibung

Die Erfindung beschriebt Verfahren und Methoden zur Verwaltung und zum Transfer von Daten innerhalb von mehrdimensionalen Anordnungen von Sendern und Empfängern. Das Aufteilen eines Datenstromes in mehrere unabhängige Zweige und das nachfolgende Zusammenfassen der einzelnen Zweige zu einem Datenstrom soll einfach durchführbar sein, wobei die einzelnen Datenströme in der korrekten zeitlichen Reihenfolge wieder zusammengefasst werden. Insbesondere zur Abarbeitung von reentrantem Code ist dieses Verfahren von besonderer Wichtigkeit. Das beschriebene Verfahren ist insbesondere für konfigurierbare Architekturen geeignet, die effiziente Steuerung der Konfiguration und Rekonfiguration findet besondere Beachtung.

Aufgabe der Erfindung ist es, Neues für die gewerbliche Nutzung bereitzustellen.

Die Lösung der Aufgabe wird unabhängig beansprucht. Bevorzugte Ausführungsformen befinden sich in den Unteransprüchen.

Unter einer rekonfigurierbaren Architektur werden vorliegend Bausteine (VPU) mit konfigurierbarer Funktion und/oder Vernetzung verstanden, insbesondere integrierte Bausteine mit einer Mehrzahl von ein- oder mehrdimensional angeordneten arithmetischen und/oder logischen und/oder analogen und/oder speichernden und/oder intern/extern vernetzenden Baugruppen, die direkt oder durch ein Bussystem miteinander verbunden sind.

Zur Gattung dieser Bausteine zählen insbesondere systolische Arrays, neuronale Netze, Mehrprozessor Systeme, Prozessoren mit mehreren Rechenwerken und/oder logischen Zellen und/oder kommunikativen/peripheren Zellen (IO), Vernetzungs- und Netzwerkbausteine wie z.B. Crossbar-Schalter, ebenso wie bekannte Bausteine der Gattung FPGA, DPGA, Chameleon, XPUTER, etc.. Hingewiesen wird insbesondere in diesem Zusammenhang auf die folgenden Schutzrechte und Anmeldungen desselben Anmelders: P 44 16 881.0-53, DE 197 81 412.3, DE 197 81 483.2, DE 196 54 846.2-53, DE 196 54 593.5-53, DE 197 04 044.6-53, DE 198 80 129.7, DE 198 61 088.2-53, DE 199 80 312.9, PCT/DE 00/01869, DE 100 36 627.9-33, DE 100 28 397.7, DE 101 10 530.4, DE 101 11 014.6, PCT/EP 00/10516, EP 01 102 674.7, PACT02, PACT04, PACT05, PACT08, PACT10, PACT11, PACT13, PACT21, PACT13, PACT15b, PACT18(a), PACT25(a,b). Diese sind hiermit zu Offenbarungszwecken vollumfänglich eingegliedert.

Die o.g. Architektur wird beispielhaft zur Verdeutlichung herangezogen und im folgenden VPU genannt. Die Architektur besteht aus beliebigen arithmetischen, logischen (auch Spei-

cher) und/oder Speicherzellen und/oder Vernetzungszellen und/oder kommunikativen/peripheren (IO) Zellen (PAEs), die zu einer ein- oder mehrdimensionalen Matrix (PA) angeordnet sein können, wobei die Matrix unterschiedliche beliebig ausgestaltete Zellen aufweisen kann, auch die Bussysteme werden dabei als Zellen verstanden. Der Matrix als ganzes oder Teilen davon zugeordnet ist eine Konfigurationseinheit (CT), die die Vernetzung und Funktion des PA beeinflußt.

Beschreibung der Erfindung

Die konfigurierbaren Zellen einer VPU müssen zur korrekten Verarbeitung von Daten miteinander synchronisiert werden. Dazu dienen zwei unterschiedliche Protokolle, eines zur Synchronisation des Datenverkehrs und ein weiteres zur Ablaufsteuerung der Datenverarbeitung. Daten werden bevorzugt über eine Mehrzahl konfigurierbarer Bussysteme übertragen. Konfigurierbare Bussysteme bedeutet insbesondere, dass beliebige PAEs Daten senden und die Verbindung zu den Empfänger-PAEs, sowie insbesondere die Empfänger-PAEs beliebig konfigurierbar sind.

Die Synchronisation des Datenverkehrs erfolgt bevorzugt durch Handshake-Protokolle, die mit den Daten übertragen werden. In der nachfolgenden Beschreibung werden einfache Handshakes sowie komplexe Verfahren beschrieben, deren bevorzugte Anwendung von der jeweilig auszuführenden Applikation oder Applikationsmenge abhängig ist.

Die Ablaufsteuerung erfolgt durch Signale (Trigger) die den Status einer PAE anzeigen. Trigger können unabhängig von den Daten über frei konfigurierbare Bussysteme geführt werden, d.h. können unterschiedliche Sender und/oder Empfänger besit-

zen und weisen bevorzugt ebenfalls Handshake-Protokolle auf. Trigger werden durch einen Status einer sendenen PAE generiert (z.B. Zero-Flag, Overflow-Flag, Negativ-Flag), indem einzelne Zustände oder Kombinationen weitergeleitet werden.

Datenverarbeitende Zellen (PAEs) innerhalb einer VPU können verscheidene Verarbeitungszustände annehmen, die vom Konfigurationszustand der Zellen und/oder eintreffenden oder eingetroffenen Triggern abhängen:

„not configured":

Keine Datenverarbeitung

„configured":

GO alle eintreffenden Daten werden berechnet.

STOP eintreffende Daten werden nicht berechnet.

STEP genau eine Berechnung wird durchgeführt.

GO, STOP und STEP werden durch im folgenden beschriebene Trigger ausgelöst.

Handshake Synchronisation

Ein besonders einfaches und dennoch sehr leistungsfähiges Handshake-Protokoll, das bevorzugt bei der Übertragung von Daten und Triggern Anwendung findet, wird nachfolgend beschrieben. Die Steuerung der Handshake-Protokolle ist bevorzugt fest in der Hardware vorgegeben und kann einen wesentlichen Bestandteil des Datenverarbeitungparadigmas einer VPU darstellen. Die Grundlagen dieses Protokolles sind bereits in PACT02 beschrieben.

Mit jeder von einem Sender über einen beliebigen Bus gesendeten Information wird ein RDY-Signal mitgesendet, das die Gültigkeit der Information anzeigt.

Der Empfänger verarbeitet nur Informationen die mit einem RDY-Signal versehen sind, alle anderen Informationen werden ignoriert.

Sobald die Informationen vom Empfänger verarbeitet wurden und der Empfänger neue Informationen entgegennehmen kann, zeigt er durch senden eines Quittierungssignales (ACK) dem Sender an, daß dieser nehe Informationen absenden darf. Der Sender wartet immer auf das Eintreffen von ACK, bevor er erneut Daten absendet.

Es wird zwischen zwei Betriebsarten unterschieden:

a) „dependend": Alle Eingänge, die Informationen entgegennehmen müssen ein gültiges RDY besitzen bevor die Information verarbeitet wird. Erst dann wird das ACK generiert.

b) „independent": Sobald ein Eingang, der Informationen entgegennimmt ein gültiges RDY besitzt, wird für diesen bestimmten Eingang ein ACK generiert, sofern der Eingang Daten entgegennehmen kann, d.h. die vorhergegangenen Daten verarbeitet wurden; ansonsten wird auf die Verarbeitung der Daten gewartet.

Die Durchführung der Synchronisation und die Steuerung der Datenverarbeitung kann nach dem Stand der Technik durch eine fest implementierte Statemachine (siehe PACT02), durch eine feingranular konfigurierte Zustandsmaschine (siehe PACT01, PACT04) oder vorzugsweise durch einen programmierbaren Sequenzer (PACT13) erfolgen. Die programmierbare Statemachine wird entsprechend des auszuführenden Ablaufes konfiguriert. Der Baustein EPS448 von Altera (ALTERA Data Book 1993) realisiert beispielsweise einen derartigen programmierbaren Sequenzer.

Eine besondere Aufgabe von Handshake-Protokolle für VPUs ist die Durchführung einer pipeline-artigen Datenverarbeitung, bei welcher insbesondere in jedem Taktzyklus Daten in jeder

PAE verabeitet werden können. Diese Forderung führt zu besonderen Ansprüchen an die Funktionsweise von Handshakes. Am Beispiel eines RDY/ACK Protokolls wird die Problemstellung und Lösung dieser Aufgabe aufgezeigt:

Figur 1a zeigt einen Aufbau einer Pipeline innerhalb einer VPU. Die Daten werden über (vorzugsweise konfigurierbare) Bussysteme (0107, 0108, 0109) an Register (0101, 0104) geführt, denen eine ggf. datenverarbeitende Logik (0102, 0105) nachgeschaltet ist. Dieser ist eine Ausgabestufe (0103, 0106) zugeordnet, die bevorzugt wieder ein Register enthält, um die Ergebnisse wieder auf einen Bus aufzuschalten. Sowohl über die Bussysteme (0107, 0108, 0109) als auch durch die datenverarbeitende Logik wird bevorzugt (0102, 0105) das RDY/ACK Protokoll zur Synchronisation übertragen.
Zwei Semantiken kommen für das RDY/ACK Protokoll in Frage:
a) ACK bedeutet "Empfänger wird Daten übernehmen", mit dem Effekt, dass die Pipeline in jedem Takt arbeitet. Es entsteht jedoch das Problem, dass aufgrund der hardwaretechnischen Realisierung im Fall eines Pipeline-Stall das ACK asynchron über alle angehaltenen Stufen der Pipeline läuft. Dies führt zu erheblichen Problemen beim Zeitverhalten, insbesondere bei großen VPUs und/oder hohen Taktfrequenzen.
b) ACK bedeutet "Empfänger hat Daten übernommen", mit dem Effekt, dass das ACK immer jeweils nur bis zur nächsten Stufe läuft und dort ein Register vorhanden ist. Das dadurch entstehende Problem ist, dass die Pipeline aufgrund der Verzögerung des in der hardwaretechnischen Realisierung notwendigen Registers nur noch in jedem zweiten Takt arbeitet.

Die Lösung der Aufgabe liegt in der Kombination beider Semantiken wie in Figur 1b dargestellt, die die Stufen 0101 bis 0103 ausschnittsweise verdeutlicht. Auf den Bussystemen

(0107, 0108, 0109) wird das Protokoll b) verwendet, indem ein Register (0110) das eintreffende RDY mit dem Einschreiben der übertragenen Daten in ein Eingangsregister um einen Takt verzögert und als ACK wieder auf den Bus weiterleitet. Diese Stufe (0110) arbeitet quasi als Protokollkonverter zwischen einem Busprotokoll und dem Protokoll innerhalb einer datenverarbeitenden Logik.

Die datenverarbeitende Logik verwendet das Protokoll a). Dieses wird erzeugt durch einen nachgeschalteten Protokollkonverter (0111). Das besondere an 0111 ist, dass eine Voraussage getroffen werden muss, ob die eintreffenden Daten von der datenverarbeitenden Logik auch tatsächlich vom Bussystem abgenommen werden. Dies wird dadurch gelöst, dass ein zusätzliches Pufferregister (0112) in den Ausgangsstufen (0103, 0106) für die auf das Bussystem zu übertragenden Daten eingeführt wird. Die durch die datenverarbeitende Logik generierten Daten werden zugleich an das Bussystem und in das Pufferregister geschrieben. Kann der Bus die Daten nicht abnehmen, d.h. das ACK des Bussystems bleibt aus, sind die Daten im uPfferregister vorhanden und werden sobald das Bussystem bereit ist über einen Multiplexer (0113) auf das Bussystem geschaltet. Ist das Bussystem sofort zur Abnahme der Daten bereit, werden die Daten über Multiplexer (0113) den direkt auf den Bus weitergeleitet. Das Pufferregister ermöglicht die Quittierung mit der Semantik a), da so lange mit "Empfänger wird Daten übernehmen" quittiert werden kann, wie das Pufferregister leer ist, da durch das Einschreiben in das Pufferregister sichergestellt ist, dass die Daten nicht verloren gehen.

Trigger

In VPU Bausteinen werden zur Übertragung einfacher Informationen sogenannte Trigger verwendet, deren Grundlagen in

PACT08 beschrieben sind. Trigger werden mittels eines in Segmente aufgeteilten ein- oder mehrdimensionalen Bussystems übertragen. Die einzelnen Segmente können mit Treibern zur Verbesserung der Signalqualität ausgestattet sein. Die jeweiligen Triggerverbindungen, die über eine Verschaltung mehrere Segmente realisiert wird, wird vom Anwender programmiert und über die CT konfiguriert.

Trigger übertragen beispielsweise vor allem, jedoch nicht ausschließlich, die nachfolgenden Informationen oder beliebige Kombinationen dieser

* Statusinformationen von Rechenwerken (ALUs), z.B.
  - Carry
  - Division by Zero
  - Zero
  - Negativ
  - Under-/Overflow
* Ergebnisse von Vergleichen und/oder Schleifen
* n-bit Informationen (für kleine n)
* Interruptanforderungen, die intern oder extern generiert werden

Trigger werden von beliebigen Zellen generiert und werden durch beliebige Ereignisse in den einzelnen Zellen angesteuert. Insbesondere können Trigger von einer CT oder einer externen Einheit, die außerhalb des Zellarrays oder des Bausteines generiert werden.

Trigger werden von beliebigen Zellen empfangen und auf beliebige Art ausgewertet. Insbesondere können Trigger von einer CT oder einer externen Einheit, die außerhalb des Zellarrays oder des Bausteines ausgewertet werden.

Trigger dienen hauptsächlich der Ablaufsteuerung innerhalb einer VPU, beispielsweise für Vergleiche und/oder Schleifen.

Datenpfade und/oder Verzweigungen können durch Trigger freigeschaltet (enabled) oder gesperrt (disabled) werden.
Ein weiterer wesentlicher Einsatzbereich von Triggern ist das Synchronisieren und Ansteuern von Sequenzern, sowie deren Informationsaustausch; ebenso wie die Steuerung der Datenverarbeitung in den Zellen.
Die Verwaltung von Triggern und die Steuerung der Datenverarbeitung kann nach dem Stand der Technik durch eine fest implementierte Statemachine (siehe PACT02, PACT08), durch eine feingranular konfigurierte Zustandsmaschine (siehe PACT01, PACT04, PACT08, [Chameleon]) oder vorzugsweise durch eine programmierbare Statemachine (PACT13) erfolgen. Die programmierbare Statemachine wird entsprechend des auszuführenden Ablaufes konfiguriert. Der Baustein EPS448 von Altera [ALTERA Data Book 1993] realisiert beispielsweise einen derartigen programmierbaren Sequenzer.

Grundverfahren

Die einfachen Synchronisationsverfahren mit Rdy/Ack Protokollen erschweren die Verarbeitung von komplizierten Datenströmen, da der Aufwand zur Einhaltung der korrekten Reihenfolge sehr hoch ist. Die korrekte Implementierung ist Aufgabe des Programmierers. Weiterhin werden zusätzliche Ressourcen für die Implementierung benötigt.
Im Folgenden wird ein einfaches Verfahren beschrieben, das diese Aufgabe löst.

Übertragung 1:n

Dieser Fall ist trivial: Der Sender schreibt die Daten auf den Bus. Die Daten liegen so lange stabil am Bus an, bis das ACK als Quittierung von allen Empfängern eingetroffen ist (die Daten „stehen"). RDY wird gepulst, d.h. liegt genau für

einen Takt an, damit die Daten nicht fälschlicherweise mehrfach gelesen werden. Sofern das RDY implementierungsabhängig Multiplexer und/oder Tore und/oder weitere geeignete Übertragungselemente ansteuert, die den Datentransfer steuern, wird diese Ansteuerung für den Zeitraum der Datenübertragung gespeichert (RdyHold). Dies bewirkt dass die Stellung der Tore und/oder Multiplexer und/oder weitere geeignete Übertragungselemente auch nach dem RDY-Puls gültig bleibt und damit weiterhin gültige Daten auf dem Bus anliegen.

Sobald ein Empfänger die Daten übernommen hat, quittiert er mittels eines ACK (vgl. PACT02). Es soll nochmals erwähnt werden, dass die korrekten Daten bis zur Abnahme durch den oder die Empfänger auf dem Bus anliegen. ACK wird vorzugsweise ebenfalls als Puls übertragen. Durchläuft ein ACK einen Multiplexer und/oder Tor und/oder ein weiteres geeignetes Übertragungselement, in welchen das RDY zuvor zur Speicherung der Ansteuerung diente (s. RdyHold), wird diese Ansteuerung nunmehr gelöscht.

Zur Übertragung von 1:n ist es sinnvoll das das ACK so lange zu halten, d.h. kein gepulstes ACK zu verwenden, bis ein neues RDY eintrifft, d.h. das ACK „steht" ebenfalls. An jedem Busknoten, welcher eine Verzweigung zu mehreren Empfängern darstellt, werden die eintreffenden ACKs miteinander verundet (AND). Da die ACKs „stehen" ergibt sich letzendlich ein „stehendes" ACK beim Sender, das die ACKs aller Empfänger repräsentiert. Um die Laufzeit der ACK-Kette durch die UND-Gatter möglichst gering zu halten, wird empfohlen eine baumförmige Busstruktur zu wählen, bzw. während des Routings des abzuarbeitenden Programmes zu erzeugen.

Stehende ACKs können nunmehr implementierungsabhängig zu dem Problem führen, dass RDY Signale geACKt werden, für die eigentlich kein ACK vorlag, da ein altes ACK zu lange stand. Eine Lösung dafür ist, ACK grundsätzlich zu pulsen und das

eintreffende ACK jedes Zweiges an einer Verzweigung zu speichern. Erst wenn die ACKs aller Zweige eintrafen wird ein ACK-Puls in Richtung Sender weitergeleitet und gleichzeitig alle gespeicherten ACKs (AckHold) und ggf. die RdyHold gelöscht.

Figur 1c zeigt die Grundlagen des Verfahrens. Ein Sender 0120 versendt Daten über ein Bussystem 0121 zusammen mit einem RDY 0122. Mehrere Empfänger (0123, 0124, 0125, 0126) erhalten die Daten und das dazugehörenden RDY (0122). Jeder Empfänger generiert ein ACK (0127, 0128, 0129, 0130), die jeweils über eine geeignete boolsche Logik (0131, 0132, 0133) z.B. eine logische UND-Funktion verknüpft und an den Sender geleitet werden (0134).

Figur 1c zeigt eine mögliche bevorzugte Ausgestaltung mit 2 Empfängern (a, b). Eine Ausgabestufe (0103) versendet Daten und das zugeordnete in diesem Beispiel gepulste RDY (0131). RdyHold-Stufen (0130) vor den Ziel-PAEs übersetzen das gepulste RDY in ein stehendes RDY. Ein stehendes RDY soll in diesem Beispiel den boolschen Wert b'1 aufweisen. Die Inhalte sämtlicher RdyHold-Stufen werden über eine Kette logischer ODER-Funktionen (0133) an 0103 zurückgeführt. Bestätigt eine Ziel-PAE die Annahme der Daten, wird lediglich durch das eintreffende ACK (0134) die jeweils entsprechende RdyHold-Stufe zurückgesetzt. Die Semantik des zurückgeführten Signals lautet somit, b'1 = "irgendeine PAE hat die Daten nicht abgenommen". Sobald alle RdyHold-Stufen zurückgesetzt sind, gelangt über die ODER-Kette (0133) die Information b'0 = "alle PAEs haben die Daten abgenommen" an 0103, was als ACK gewertet wird. Die Ausgänge (0132) der RdyHold-Stufen können wie bereits beschrieben zur Ansteuerung von Busschaltern mitverwendet werden.

Dem letzten Eingang einer ODER-Kette wird ein logischen b'0 aufgeschaltet, damit die Kette ordentlich funktioniert.

Übertragung n:1

Dieser Fall ist vergleichsweise komplex. (F1) Einerseits müssen mehrere Sender auf einen Empfänger gemultiplext werden, (F2) andererseits muß zumeist auch die zeitliche Reihenfolge der Sendungen eingehalten werden. Nachfolgend werden mehrere Verfahren zur Lösung der Aufgabe beschrieben. Es soll darauf hingewiesen werden, daß kein Verfahren grundsätzlich zu bevorzugen ist. Vielmehr ist je nach System und auszuführenden Algorithmen das jeweils aus Sicht der Programmierbarkeit, des Aufwands und der Kosten am besten Geeignete zu wählten.

Eine einfache n:1 Übertragung kann dadurch realisiert werden, dass jeweils mehrere Datenpfade auf die Eingänge von PAEs geführt werden. Die PAEs werden als Multiplexerstufen konfiguriert. Eingehende Trigger steuern den Multiplexer und selektieren jeweils einen der Mehrzahl von Datenpfaden. Sofern erforderlich, können Baumstrukturen aus als Multiplexer konfigurierten PAEs aufgebaut werden um eine Vielzahl von Datenströme (grosse n) zusammenzuführen. Das Verfahren erfordet die besondere Aufmerksamkeit des Programmierers um die unterschiedlichen Datenströme zeitlich korrekt zu sortieren. Insbesondere sollten alle Datenpfade dieselbe Länge und/oder Verzögerung aufweisen, um die korrekte Reihenfolge der Daten zu gewährleisten.

Leistungsfähigere Zusammenführungsmethoden werden nachfolgend beschieben:
Da F1 zunächst trivial durch einen beliebigen Arbiter mit nachgeschaltetem Multiplexer lösbar scheint, soll die Überlegung mit F2 begonnen werden.

Die Einhaltung der zeitlichen Reihenfolge ist mit einfachen Arbitern nicht möglich. Figur 2 zeigt ein erstes mögliches Implementierungbeispiel. Ein FIFO (0206) wird verwendet, um die zeitliche Reihenfolgen von Übertragungsanforderungen auf ein Bussystem (0208) korrekt abzulegen und abzuarbeiten. Hierzu wird jedem Sender (0201, 0202, 0203, 0204) eine eindeutige Nummer zugeordnet, die seine Adresse darstellt. Jeder Sender fordert eine Datenübertragung auf das Bussystem 0208 an, indem er seine Adresse auf einem Bus (0209, 0210, 0211, 0212) anzeigt. Die jeweiligen Adressen werden über einen Multiplexer (0205) in einem FIFO (0206) entsprechend der Reihenfolge der Sendeanforderungen gespeichert. Das FIFO wird schrittweise abgearbeitet und die Adresse des jeweiligen FIFO-Eintrages wird auf einem weiteren Bus (0207) angezeigt. Dieser Bus adressiert die Sender und der Sender mit der entsprechenden passenden Adresse erhält den Zugriff auf den Bus 0208. Für ein derartiges Verfahren können beispielsweise die internen Speicher der VPU Technologie als FIFO verwendet werden (vgl. PACT04, PACT13).

Bei näherer Betrachtung entsteht jedoch folgendes Problem: Sobald mehrer Sender gleichzeitig auf den Bus zugreifen wollen, muß ein Sender ausgewählt werden, dessen Adresse dann in das FIFO gespeichert wird. Im nächsten Takt wird dann der nächste Sender ausgewählt, usw. Die Auswahl kann durch einen Arbiter (0205) erfolgen. Dadurch wird die Gleichzeitigkeit aufgelöst, was jedoch zumeist kein Problem darstellt. Für Realtime-Anwendungen könnte ein priorisierender Arbiter verwendet werden. Das Verfahren scheitert jedoch an einer einfachen Überlegung: Zum Zeitpunkt t fordern drei Sender S1,S2 und S3 den Empfänger E an. Bei t wird S1, bei t+1 wird S2 und bei t+2 wird S3 gespeichert. Bei t+1 fordert jedoch S4 und

S5, bei t+2 zusätzlich S6 und wieder S1 den Empfänger an. Da nun neue Anforderungen mit Alten überlappen, wird die Abarbeitung sehr schnell äußerst komplex und erfordert einen erheblichen zusätzlichen Hardwareaufwand.

Somit ist das in Figur 2 beschriebene Verfahren bevorzugt für einfache n:1 Übergänge die nach Möglichkeit keine gleichzeitigen Busanforderungen aufweisen anzuwenden.

Nach dieser Überlegung scheint es sinnvoll nicht einen Sender je Takt zu speichern, sondern die Menge aller Sender die zu einem bestimmten Takt die Übertragung anfordern. Zum jeweils darauffolgenden Takt wird dann die jeweils neue Menge gespeichert. Sofern mehrere Sender zum gleichen Takt die Übertragung anfordern, werden diese bei der Abarbeitung des Speichers arbitriert.

Das Speichern mehrerer Senderadressen zugleich ist jedoch ebenfalls sehr aufwendig. Eine einfache Realisierung wird durch folgende Ausführung in Figur 3 erreicht:

- Ein zusätzlicher Zähler (REQCNT, 0301) zählt die Zahl der Takte T. Jeder Sender (0201, 0202, 0203, 0204), der beim Takt t die Übertragung anfordert, speichert den Wert von REQCNT (REQCNT(t)) beim Takt t als seine Adresse.
- Jeder Sender, der beim Takt t+1 die Übertragung anfordert speichert, den Wert von REQCNT (REQCNT(t+1)) beim Takt t+1 als seine Adresse.
- ...
- Jeder Sender, der beim Takt t+n die Übertragung anfordert speichert, den Wert von REQCNT (REQCNT(t+n)) beim Takt t+n als seine Adresse.

Das FIFO (0206) speichert nunmehr die Werte von REQCNT(tb) bei einem bestimmten Takt tb.

Das FIFO zeigt einen gespeicherten Wert von REQCNT als Sendeaufforderung auf einem separaten Bus (0207) an. Jeder Sender vergleicht diesen Wert mit dem von ihm gespeicherten. Sind die Werte gleich sendet er die Daten. Besitzen mehrere Sender denselben Wert, d.h. wollen gleichzeitig Daten übertragen, so wird die Übertragung jetzt mittels eines geeigneten Arbiters (CHNARB, 0302b) arbitriert und mittels eines durch den Arbiter angesteuerten Multiplexers (0302a) auf den Bus geschaltet. Eine beispielsweise mögliche Ausgestaltung des Arbiters wird nachfolgend beschrieben.

Sobald keine Sender mehr auf einen REQCNT Wert ansprechen, d.h. dem Arbiter liegen keine Busanforderungen zur Arbitrierung mehr vor (0303) , schaltet das FIFO zum nächsten Wert weiter. Beinhaltet das FIFO keine gültigen Einträge mehr (empty), werden die Werte als ungültig gekennzeichnet, damit keine fälschlichen Buszugriffe entstehen.

In einer bevorzugten Ausgestaltung werden nur die Werte von REQCNT in den FIFO (0206) gespeichert bei denen eine Busanforderung eines Senders (0201, 0202, 0203, 0204) vorlag. Dazu signalisiert jeder Sender seine Busanforderung (0310, 0311, 0312, 0313). Dieser werden logisch verknüpft (0314), z.B. durch eine ODER-Funktion. Die entstehende Sendeanforderung aller Sender (0315) auf ein Tor (0316) geführt, das nur die Werte von REQCNT an das FIFO (0206) leitet, an bei denen tatsächlich eine Busanforderung vorlag.

Das beschriebene Verfahren kann ein einer bevorzugten Ausführung entsprechend Figur 4 wie folgt weiter optimiert werden: Durch REQCNT (0410) wird eine lineare Folge von Werten (REQCNT(tb)) generiert, wenn statt aller Takte t nur die Takte gezählt werden, in denen eine Busanforderung eines Senders

(0315) existiert. Durch die von REQCNT generierte nun lückenlose lineare Folge von Werten ist das FIFO durch einen einfachen Zähler (SNDCNT, 0402) ersetzbar, der ebenfalls linear zählt und dessen Wert (0403) entsprechend 0207 die jeweiligen Sender freischaltet. Dabei zählt SNDCNT weiter sobald kein Sender mehr auf den Wert von SNDCNT reagiert. Sobald der Wert von REQCNT gleich dem Wert von SNDCNT ist, zählt SNDCNT nicht mehr weiter, da der letzte Wert erreicht ist.

Für sämtliche Implementierungsarten gilt, daß die maximal erforderliche Breite von REQCNT gleich $\log_2$(Anzahl_der_Sender) ist. Bei Überschreiten des größtmöglichen Wertes beginnt REQCNT und SNDCNT wieder beim minimalen Wert (für gewöhnlich 0).

Arbiter

Eine Vielzahl von Arbitern sind nach dem Stand der Technik als CHNARB einsetzbar. Je nach Anwendung eignen sich besser priorisierte oder unpriorisierte Arbiter, wobei priorisierte den Vorteil bieten, daß sie bei Realtime-Aufgaben bestimmte Tasks bevorzugen können.
Im folgenden wird ein serieller Arbiter beschrieben, der in der VPU-Technologie besonders einfach und resourcensparend implementierbar ist. Zudem bietet der Arbiter den Vorteil priorisierend zu arbeiten, wodurch die bevorzugte Bearbeitung bestimmter Übertragungen ermöglicht wird.

Zunächst wird ein möglicher Grundaufbau eines Bussystems in Figur 5 beschrieben. Bausteine der Gattung VPU besitzen ein Netzwerk aus parallelen Daten-Bussystemen (0502), wobei jede PAE zur Datenübertragung mindestens Anschluß an einen Datenbus besitzt. Für gewöhnlich ist ein Netzwerk aus mehreren gleichwertigen parallelen Datenbussen (0502) aufgebaut, wobei

je ein Datenbus für eine Datenübertragung konfiguriert sein kann. Die verbleibenden Datenbusse können für andere Datenübertragungen zur freien Verfügung stehen.

Es soll weiterhin erwähnt werden, daß die Datenbusse segmentierbar sein können, d.h. durch Konfiguration (0521) kann ein Bussegment (0502) über Tore (G) auf das benachbarte Bussegment (0522) durchgeschaltet werden. Die Tore (G) können aus Transmission-Gates aufgebaut sein und bevorzugt Signalverstärker und/oder Register aufweisen.

Eine PAE (0501) greift vorzugsweise über Multiplexer (0503) oder eine vergleichbare Schaltung Daten von einem der Busse (0502) ab. Die Freischaltung der Multiplexeranordung ist konfigurierbar (0504).
Bevorzugt werden die von einer PAE (0510) generierten Daten (Ergebnisse) über eine ähnliche unabhängig konfigurierbare (0505) Multiplexer-Schaltung auf einen Bus (0502) aufgeschaltet.
Die in Figur 5 beschriebene Schaltung wird mit Busknoten bezeichnet.
Ein einfacher Arbiter für einen Busknoten kann folgendermassen, wie in Figur 6 dargestellt, implementiert sein:

Durch zwei AND-Gatter (0601, 0602) kann das Grundelement 0610 eines einfachen seriellen Arbiters aufgebaut werden Figur 6a. Das Grundelement besitzt einen Eingang (RDY, 0603) durch den ein Eingangsbus anzeigt, dass er Daten überträgt und eine Freischaltung auf den Empfängerbus anfordert. Ein weiterer Eingang (ACTIVATE, 0604) der in diesem Beispiel durch einen logischen 1-Pegel anzeigt, daß keines der vorhergehenden Grundelemente aktuell den Bus arbitriert hat und somit eine Arbitrierung durch dieses Grundelement zulässig ist. Der Aus-

gang RDY_OUT (0605) zeigt beispielsweise einem nachgeschalteten Busknoten an, daß das Grundelement den Buszugriff freischaltet (wenn eine Busanforderung (RDY) besteht) und ACTIVATE_OUT (0606) zeigt an, daß das Grundelement aktuell keine Freischaltung (mehr) durchführt, da keine Busanforderung (RDY) (mehr) besteht und/oder keine vorherige Arbiterstufe den Empfängerbus (ACTIVE) belegt hat.

Durch die serielle Verkettung entsprechend Figur 6b von ACTIVATE und ACTIVATE_OUT über die Grundelemente 0610 entsteht ein serieller priorisierender Arbiter, wobei das erste Grundelement die höchste Priorität besitzt und dessen ACTIVATE Eingang immer aktiviert ist.

Durch das bereits beschriebene Protokoll ist sichergestellt, daß innerhalb desselben SNDCNT-Wertes jede PAE nur eine Datenübertragung durchführt, da eine nachfolgende Datenübertragung einen anderen SDNCNT-Wert besitzen würde. Diese Bedingung ist für eine störungsfreie Funktion der seriellen Arbiters erforderlich, da damit die für die Priorisierung notwendige Abarbeitungsreihenfolge der Freischaltungsanforderungen (RDY) gewährleistet wird. Mit anderen Worten, kann eine Freigabeanforderung (RDY) nicht während einer Arbitrierung nachträglich an Grundelementen auftreten, die bereits durch ACTIVATE_OUT anzeigen, daß sie keine Freischaltung eines Buszugriffs ermöglichen.

Lokalität und Laufzeit

Grundsätzlich ist das Verfahren über lange Strecken hinweg einsetzbar. Ab einer von der Systemfrequenz abhängigen Länge ist die Übertragung der Daten und Ausführung des Protokolls nicht mehr in einem Takt möglich.

Eine Lösung ist die Datenpfade exakt gleich lang auszulegen und die Zusammenführung an genau einer Stelle durchzuführen. Damit sind sämtliche Steuersignale für das Protokoll lokal, wodurch eine Erhöhung der Systemfrequenz möglich wird. Zum Ausbalancieren der Datenpfade bieten sich FIFO-Stufen an, die als Verzögerungsstufen (delayline) mit konfigurierbarer Verzögerund arbeiten, die nachfolgend genauer beschrieben werden.

Eine weitaus optimalere Lösung, bei der auch Datenpfade baumförmig zusammengeführt werden können, ist wie folgt aufbaubar:

Abgewandeltes Protokoll, Timestamp

Vorbedingung ist, daß ein Datenpfad in mehrere Zweige aufgetrennt und später wieder zusammengeführt wird. Dies geschieht gewöhnlicherweise bei Verzweigungen wie bei den Programmierkonstrukten „IF" oder „CASE". In Figur 7a ist ein CASE-ähnliches Konstrukt beispielhaft dargestellt.

Spätestens der letzten PAE vor einer Verzweigung (0701) wird ein REQCNT (0702) zugeordnet, der jedem Datenwort einen Wert (Timestamp) zuweist, der im weiteren immer zusammen mit dem Datenwort übertragen wird. REGCNT zählt linear mit jedem Datenwort weiter, sodass durch einen eindeutigen Wert die Position eines Datenwortes innerhalb eines Datenstroms bestimmbar ist. Die Datenworte verzweigen sich nachfolgend in mehrere unterschiedliche Datenpfade (0703, 0704, 0705). Mit jedem Datenwort wird der ihm zugeordneten Wert (Timestamp) durch die Datenpfade geleitet.

Vor der/den PAE (0708) die den zusammengeführten Datenpfad weiterverarbeiten, sortiert ein Multiplexer (0707) die Datenworte wieder in die richtige Reihenfolge. Dazu ist dem Multiplexer ein linear zählender SNDCNT (0706) zugeordnet. Der Wert (Timestamp) der jedem Datenwort zugewiesen wurde, wird

mit dem Wert von SNDCNT verglichen. Das jeweils passende Datenwort wird durch den Multiplexer selektiert. Paßt zu einem bestimmten Zeitpunkt kein Datenwort, wird keine Selektion durchgeführt. SNDCNT zählt nur dann weiter, wenn ein passendes Datenwort selektiert wurde.

Um eine möglichst hohe Taktfrequenz zu erreichen ist dabei die Zusammenführung der Datenpfade sehr lokal durchzuführen. Damit werden die Leitungslängen minimiert und die damit verbundenen Laufzeiten gering gehalten.

Gegebenenfalls sind die Längen der Datenpfade durch Registerstufen (Pipelines) auszugleichen, bis sämtliche Datenpfade an einem gemeinsamen Punkt zusammengeführt werden können. Dabei sollte darauf geachtet werden, daß die Längen der Pipelines in etwa gleich sind, um keine allzu großen Zeitverschiebungen zwischen den Datenworten zu erhalten.

Verwendung der Timestamp zum Multiplexen

Der Ausgang einer PAE (PAE-S) wird an mehrere PAE (PAE-E) weitergeleitet. Nur eine der PAEs soll die Daten je Taktzyklus verarbeiten. Die PAE-E haben eine jeweils unterschiedliche fest konfigurierte Adresse, die jeweils mit dem TimeStamp-Bus verglichen wird. Die PAE-S selektiert die empfangende PAE dadurch, daß sie die Adresse der empfangenden PAE auf den TimeStamp Bus ausgibt. Dadurch wird die PAE adressiert, für die die Daten jeweils bestimmt ist.

Spekulative Ausführung und Task-Switch

Von klassischen Mikroprozessoren ist das Problem der spekulativen Ausführung bekannt. Dieses tritt auf, wenn die Verarbeitung von Daten von einem Ergebnis der vorhergehenden Datenverarbeitung abhängig ist; aber jedoch mit der Verarbei-

tung der abhängigen Daten aus Performance-Gründen jedoch schon vorab - ohne daß das erforderliche Ergebnis vorliegt - begonnen wird. Ist das Ergebnis ein anderes als vorab angenommen, muß die Verarbeitung der auf fehlerhaften Annahmen basierenden Daten neu durchgeführt werden (Fehlspekulation). Generell kann dies auch in VPUs auftreten.
Durch Umsortierung und ähnliche Verfahren kann dieses Problem minimiert werden, wobei jedoch dessen Auftreten niemals ausgeschlossen werden kann.

Ein ähnliches Problem liegt dann vor, wenn durch eine, der Datenverarbeitung in einerhalb des PAs übergeordnete, Einheit (z.B. der Task-Scheduler eines Betriebssystem, Echtzeitanforderung, usw.) die Datenverarbeitung abbricht, bevor diese vollständig ausgeführt wurde. In diesem Fall muß der Zustand der Pipeline derart gesichert werden, daß die Datenverarbeitung wieder nach der Stelle der Operanden beginnt, die zur Berechnung des letzen fertigen Ergebnisses geführt haben.

Innerhalb einer Pipeline treten zwei relevante Zustände auf:
RD Am Beginn einer Pipeline wird angezeigt, daß neue Daten angenommen oder angefordert werden.
DONE Am Ende einer Pipeline wird die korrekte Verarbeitung von Daten angezeigt, bei denen keine Fehlspekulation auftrat.
Weiterhin kann der Zustand MISS_PREDICT verwendet werden, der anzeigt, daß eine Fehlspekulation auftrat. Hilfsweise kann dieser Zustand auch durch eine Negierung des Zustandes DONE zu geeignetem Zeitpunkt generiert werden.

Spezielle FIFOs

Aus PACT04 und PACT13 sind Verfahren bekannt, bei welchen Daten in Speichern gehalten werden und aus diesen zur Verarbei-

tung ausgelesen werden, bzw. Ergebnisse in diese abgelegt werden. Dazu können mehrere unabhängige Speicher verwendet werden. Die Speicher können in unterschiedlichen Betriebsarten arbeiten, insbesondere kann ein wahlfreier Zugriff, ein Stack- oder FIFO-Betriebsmodus verwendet werden.
Daten werden in VPUs für gewöhnlich linear verarbeitet, sodass der FIFO-Betriebsmodus häufig bevorzugt zum Einsatz kommt. Es soll beispielhaft eine besondere Erweiterung der Speicher für den FIFO Betriebsmodus vorgestellt werden, der Spekulation direkt unterstützt und im Fall einer Fehlspekulation die wiederholte Verarbeitung der fehlspekulierten Daten ermöglicht. Desweiteren unterstützt der FIFO Taskswitches zu beliebigen Zeitpunkten.

Zunächst wird der erweiterte FIFO Betriebsmodi am Beispiel eines Speichers ausgeführt, auf den im Rahmen einer bestimmten Datenverarbeitung lesend (Leseseite) zugegriffen wird. Der beispielhafte FIFO ist in Figur 8 dargestellt.
Der Aufbau der Schreibschaltung entspricht mit einem gewöhnlichen Schreibzeiger (WR_PTR, 0801) dabei dem Stand der Technik, der sich mit jedem Schreibzugriff (0810) weiterbewegt. Die Leseschaltung besitzt z.B. den üblichen Zähler (RD_PTR, 0802), der entsprechend eines Lesesignals (0811) jedes gelesene Wort zählt und entsprechend die Leseadresse des Speichers (0803) modifiziert. Neu gegenüber dem Stand der Technik ist eine zusätzliche Schaltung DONE_PTR (0804), die nicht die ausgelesenen Daten dokumentiert, sondern die ausgelesenen und korrekt verarbeiteten, mit anderen Worten also nur die Daten bei denen kein Fehler auftrat und deren Ergebnis am Ende der Berechnung ausgegeben und das korrekte Berechnungsende ein Signal (0812) angezeigt wurde. Mögliche Schaltungen werden nachfolgend beschrieben.

Das FULL-Flag (0805) (nach den Stand der Technik), das anzeigt, daß der FIFO voll ist und keine weiteren Daten mehr gespeichert werden können, wird nunmehr durch einen Vergleich (0806) von DONE_PTR mit WR_WTR generiert. Dadurch wird sichergestellt, dass Daten, auf die bedingt durch eine mögliche Fehlspekulation ein Rückgriff notwendig wird, nicht überschrieben werden.

Das EMPTY-Flag (0807) wird entsprechend dem üblichen Aufbau durch Vergleich (0808) des RD_PRT mit dem WR_PTR generiert. Trat eine Fehlspekulation (MISS_PREDICT, 0809) auf, wird der Lesezeiger mit dem Wert DONE_PTR + 1 geladen. Damit wird die Datenverarbeitung wieder bei dem Wert begonnen, der eine Fehlspekulation auslöste.

Zwei mögliche Ausgestaltungen des DONE_PTR sollen beispielhaft genauers ausgeführt werden:

a) Implementierung durch einen Zähler

DONE_PTR ist als Zähler implementiert, der bei einem Reset der Schaltung oder zu Beginn einer Datenverarbeitung gleich dem RD_PTR gesetzt wird. Durch ein eingehendes Signal (DONE) wird angezeigt, daß die Daten erfolgreich, d.h. ohne Fehlspekulation verarbeitet wurden. Dadurch wird DONE_PRT derart modifiziert, daß er auf das nächste sich in Verarbeitung befindende Datenwort zeigt.

b) Implementierung durch einen Subtrahierer

Sofern die Länge der datenverarbeitenden Pipeline immer exakt bekannt ist und sichergestellt ist, daß die Länge konstant ist (also keine Verzweigung in unterschiedlich lange Pipelines stattfindet), kann ein Subtrahierer eingesetzt werden. In einem zugeordneten Register wird die Länge der Pipeline vom Anschluß des Speichers bis zum Erkennen einer möglichen Fehlspekulation gespeichert. Dadurch muß die Datenverarbeitung nach einer Fehlspekulation bei dem Datenwort wieder aufgesetzt werden, das durch die Differenz berechnet werden kann.

Auf der Schreibseite, um das Ergebnis der Datenverarbeitung einer Konfiguration zu sichern, ist ein entsprechend ausgestalteter Speicher erforderlich, wobei die Funktion des DONE_PRT für den Schreibzeiger implementiert ist, um bereits (fehl-)berechnete Ergebnisse bei einem erneuten Durchlauf der Datenverarbeitung wieder zu überschreiben. Mit anderen Worten ist die Funktion des Schreib-/Lesezeigers entsprechend der in der Zeichnung geklammerten Adressen vertauscht.

Wird die Verarbeitung der Daten von einer anderen Quelle unterbrochen (z.B. Taskswitch eines Betriebssystems) ist es hinreichend den DONE_PTR zu sichern und die Datenverarbeitung zu einem späteren Zeitpunkt bei DONE_PTR + 1 wieder aufzusetzen.

FIFOs für Input/Output-Stufen, z.B. 0101, 0103

Zum Ausbalancieren von Datenpfaden und/oder Zuständen unterschiedlicher Kanten eines Graphen, bzw. unterschiedlicher Zweige der Datenverarbeitung (Trigger, vgl. PACT08, PACT13) ist es sinnvoll konfigurierbare FIFOs an den Ausgängen oder Eingängen der PAEs einzusetzen. Die FIFOs besitzen einstellbare Latenzzeiten, sodass die Verzögerung unterschiedlicher Kanten/Zweige, also die Laufzeit von Daten über unterschiedliche aber zumeist parallele Datenpfade, aufeinander abgestimmt werden kann.

Da es innerhalb einer VPU zum Anhalten einer Pipeline aufgrund ausstehender Daten oder eines ausstehenden Triggers kommen kann, sind die FIFOs ebenfalls zum Ausgleichen derartiger Verzögerungen sinnvoll. Die nachfolgend beschriebenen FIFOs lösen beide Aufgaben:

Eine FIFO-Stufe kann beispielsweise wie in Figur 9 dargestellt folgendermassen aufgebaut sein: Einem Register (0901) ist ein Multiplexer (0902) nachgeschaltet. Das Register speichert die Daten (0903) und auch deren korrekte Existenz, d.h. das dazugehörende RDY (0904). Das Einschreiben in das Register erfolgt genau dann, wenn die benachbarte FIFO-Stufe, die näher zum Ausgang (0920) des FIFOs liegt anzeigt, dass sie voll ist (0905) und ein RDY (0904) für die Daten anliegt. Der Multiplexer leitet Eingehende Daten (0903) so lange direkt auf den Ausgang weiter (0906), bis Daten in das Register geschrieben wurden und die FIFO-Stufe damit selbst voll ist, was der benachbarten FIFO-Stufe, die näher zum Eingang (0921) des FIFOs liegt angezeigt wird (0907). Die Annahme von Daten in einer FIFO-Stufe wird mit einem Input-Acknowledge (IACK, 0908) bestätigt. Die Abnahme von Daten aus einem FIFO wird durch Output-Acknowledge (OACK, 0909) bestätigt. OACK gelangt zugleich an alle FIFO-Stufen und bewirkt das weiterschieben der Daten im FIFO um jeweils eine Stufe.

Einzelne FIFO-Stufen können zum Aufbau beliebig langer FIFOs kaskadiert werden Figur 9a. Dazu werden alle IACK-Ausgänge logisch miteinander verknüpft, beispielsweise durch eine ODER-Funktion (0910).

Die Funktionsweise wird am Beispiel von Figur 10a,b erläutert:

Hinzufügen eines Datenwortes

Ein neues Datenwort wird über die Multiplexer der einzelnen FIFO-Stufen an den Registern vorbeigeleitet. Die erste volle FIFO-Stufe (1001) signalisiert der davorliegenden Stufe (1002) anhand des gespeicherten RDY, daß sie keine Daten annehmen kann. Die davorliegende Stufe (1002) hat kein RDY ge-

speichert, kennt aber den "Voll"-Zustand der Nachfolgenden (1001). Daher speichert die Stufe die Daten und das RDY (1003); und quittiert die Speicherung durch ein ACK an den Sender. Der Multiplexer (1004) der FIFO-Stufe schaltet dabei derart um, dass er nicht mehr den Datenpfad an die nachfolgende Stufe weiterleitet, sondern den Inhalt des Registers.

Entfernen eines Datenwortes

Geht ein ACK (1011) bei der letzten FIFO-Stufe ein, werden die Daten jeder vorhergehenden Stufe an die jeweils Nachfolgende übertragen (1010). Dies geschieht durch Anlegen eines globalen Schreibtaktes an jede Stufe. Da sämtliche Multiplexer bereits entsprechend der Registerbelegung gestellt sind, rutschen somit alle Daten im FIFO um eine Zeile nach unten.

Entfernen und gleichzeitiges Hinzufügen eines Datenwortes

Liegt der globale Schreibtakt an, wird in der ersten freien Stufe kein Datenwort gespeichert. Da der Multiplexer dieser Stufe die Daten noch an die Nachfolgende weiterleitet, speichert die erste volle Stufe (1012) die Daten. Deren Daten werden wie zuvor beschrieben im selben Takt von der nachfolgenden Stufe gespeichert. Mit anderem Worten: Neu einzuschreibenden Daten rutschen automatisch in die nunmehr erste freie FIFO-Stufe (1012), d.h. die vormals letzte volle FIFO-Stufe, die durch eintreffen von ACK geleert wurde, nach.

Konfigurierbare Pipeline

Es kann für bestimmte Anwendungen von Vorteil sein, mittels eines Schalters (0930) in den beispielhaft in Figur 9 gezeigten FIFO-Stufe einzelne Multiplexer des FIFOs derart zu

schalten, daß grundsätzlich das entsprechende Register eingeschaltet ist. Damit ist eine feste Latenz- bzw. Verzügerungszeit bei der Datenübertragung einstellbar über den Schalter konfigurierbar.

Zusammenführen von Datenströmen (mergen)

Zum Zusammenführen der Datenströme stehen insgesamt 3 Verfahren zur Verfügung, die je nach Anwendung unterschiedlich gut geeignet sind:

a) Local Merge
b) Tree Merge
c) Memory Merge

Lokales Zusammenführen (Local Merge)

Die einfachste Variante ist der local merge. Dabei werden alle Datenströme bevorzugt an einem einzigen Punkt oder vergleichsweise lokal zusammengeführt und gegebenenfalls sofort wieder aufgetrennt. Ein lokaler SNDCNT selektiert über einen Multiplexer genau das Datenwort, das dessen Timestamp dem Wert von SNDCNT entspricht und daher aktuell erwartet wird. Zwei Möglichkeiten sollen anhand der Figuren 7a und 7b nähers erläutert werden:

a) Ein Zähler SNDCNT (0706) zählt bei jedem eintreffenden Datenpaket weiter. Je Datenpfad ist ein Vergleicher nachgeschaltet, der den Zählerstand jeweils mit der Timestamp des Datenpfades vergleicht. Stimmen die Werte überein, wird das aktuelle Datenpaket über den Multiplexer an die nachfolgenden PAEs weitergeleitet.

b) Die Lösung nach a) wird derart erweitert, daß nach der Auswahl des jeweils aktiven Datenpfades als Herkunftsdatenpfad diesem beispielsweise mittels einer bevorzugt über ein

Übersetzungsverfahren, z.B. eine CT konfigurierbaren Lookup-Tabelle (0710), ein Zieldatenpfad zugeordnet wird. Der Herkunftsdatenpfad wird ermittelt indem die mit den Daten eintreffenden Timestamp entsprechend Verfahren a) mit einem SNDCNT (0711) verglichen (0712) werden und der übereinstimmende Datenpfad adressiert (0714) und über einen Multiplexer (0713) selektiert wird. Die Adresse (0714) wird mittels der beispielhaften Lookup-Tabelle (0710) einer Zieldatenpfadadresse (0715) zugeordnet die über einen Demultiplexer (0716) den Zielpfad auswählt. Sofern die beschriebene Struktur in Busknoten ähnlich Figur implementiert ist, kann über die beispielhafte Lookup-Tabelle (0710) die Datenverbindung auch der dem Busknoten zugeordnete PAE (0718) aufgebaut werden, beispielsweise über eine Tor-Funktion (Transmission-Gates) (0717) zum Eingang der PAE.

Ein besonders leistungsfähiges Schaltungsbeispiel ist in Figur 7c dargestellt. Eine PAE (0720) besitzt 3 Dateneingänge (A,B,C), wie beispielsweise in der XPU128ES. Bussysteme (0733) können konfigurierbare und/oder multiplexbar und je Taktzyklus selektierbar auf die Dateneingänge aufgeschaltet werden. Jedes Bussystem überträgt Daten, Handshakes und die zugeordnete Timestamp (0721). Die Eingänge A und C der PAE (0720) werden dazu verwendet, die Timestamp der Datenkanäle an die PAE weiterzuleiten (0722, 0723). Die einzelnen Timestamp können beispielsweise durch das nachfolgend beschriebene SIMD-Bussystem gebündelt werden. Die gebündelten Timestamp werden in der PAE wieder aufgetrennt und jede Timestamp je einzeln (0725, 0726, 0727) mit einem in der PAE realisierten/konfigurierten SNDCNT (0724) verglichen (0728). Die Ergebnisse der Vergleiche werden verwendet um die Eingangsmultiplexer (0730) derart anzusteuern, dass das Bussystem mit der korrekten Timestamp auf eine Sammelschiene

(0731) durchgeschaltet wird. Die Sammelschiene ist bevorzugt mit dem Eingang B verbunden, um eine Datenweiterleitung an die PAE entsprechend 0717, 0718 zu ermöglichen. Die Ausgangsdemultiplexer (0732) zur Weiterleitung der Daten auf unterschiedliche Bussysteme werden ebenfalls durch die Ergebnisse angesteuert, wobei bevorzugt eine Umordnung der Ergebnisse durch eine flexible Übersetzung, z.B. durch eine Lookup-Tabelle (0729) stattfindet, sodass den Ergebnissen frei über Demultiplexer (0732) zu selektierende Bussysteme zugeordnet werden können.

Baumartiges Zusammenfügen (Tree Merge)

In vielen Applikationen ist es wünschenswert an mehreren Punkten Teile eines Datenstromes zu mergen. Es ergibt sich daraus eine Baum ähnliche Struktur. Dabei entsteht das Problem, daß keine zentrale Entscheidung über die Selektion eines Datenwortes gefällt werden kann, sondern daß die Entscheidung über mehrere Knoten verteilt ist. Daher ist erforderlich an alle Knoten den jeweiligen Wert von SNDCNT zu übertragen. Bei hohen Taktfrequenzen ist das jedoch nur mit einer Latenzzeit, die beispielsweise durch mehrere Registerstufen während der Übertragung entsteht, möglich. Damit bietet diese Lösung zunächst keine sinnvolle Performance.
Ein Verfahren zum Verbessern der Performance ist, daß lokale Entscheidungen in jedem Knoten unabhängig vom Wert von SNDCNT zugelassen werden. Ein beispielsweise einfacher Ansatz ist, das Datenwort mit der jeweils kleinsten Timestamp an einem Knoten zu selektieren. Dieser Ansatz wird jedoch problematisch, wenn ein Datenpfad an einem Knoten zu einem Takt kein Datenwort liefert. Nun kann nicht entschieden werden, welcher Datenpfad der zu bevorzugende ist.

Der nachfolgende Algorithmus verbessert dieses Verhalten:

a) Jeder Knoten erhält einen eigenständigen SNDCNT Zähler $SNDCNT_K$.

b) Jeder Knoten soll n Eingangsdatenpfade ($P_0..P_n$) besitzen.

c) Jeder Knoten kann mehrere Ausgangsdatenpfade besitzen, die mittels eines Übersetzungsverfahrens, z.B. einer durch eine übergeordnete Konfigurationseinheit CT konfigurierbare Lookup-Tabelle, abhängig vom Eingangsdatenpfad selektiert werden.

d) Der Root-Knoten besitzt den Haupt-SNDCNT auf den alle $SNDCNT_K$ gegebenenfalls synchronisiert werdem.

Zur Selektion des korrekten Datenpfades wird folgender Algorithmus angewendet:

I. Sofern Daten an allen P_n Eingangsdatenpfaden anstehen gilt:

a) Auswahl des Datenpfades $P_{(Ts)}$ mit der kleinsten Timestamp Ts.

b) Zuweisung von K := Ts+1; SNDCNT > Ts+1 gilt $SNDCNT_K$:= SNDCNT.

II. Wenn nicht an allen Pn Eingangsdatenpfaden Daten anstehen gilt:

a) Auswahl eines Datenpfades nur, wenn die Timestamp Ts == $SNDCNT_K$

b) $SNDCNT_K$:= SNDCNT + 1

c) SNDCNT := SNDCNT + 1

III. Wird in einem Takt keine Zuweisung durchgeführt gilt:

a) $SNDCNT_S$:= SNDCNT

IV. Der Root-Knoten besitzt den SNDCNT, der bei jeder Selektion eines gültigen Datenwortes weiterzählt und die korrekte Reihenfolge der Datenwörter bei der Root des Trees sicher-

stellt. Alle andern Knoten werden auf Wert von SNDCNT synchronisiert, sofern erforderlich (siehe 1-3). Dabei tritt eine Latenzzeit auf, die der Anzahl der Register entspricht, die zu Überbrückung der Strecke von SNDCNT nach $SNDCNT_K$ eingefügt werden müssen.

Figur 11 zeigt einen möglichen Baum, der beispielsweise auf PAEs ähnlich denen der VPU XPU128ES aufgebaut sind. Ein Root-Knoten (1101) besitzt einen integrierten SNDCNT, dessen Wert am Ausgang H (1102) zur Verfügung steht. Die Datenworte an den Eingänge A und C werden entsprechend des beschriebenen Verfahrens selektiert und jeweils das Datenwort mit der korrekten Reihenfolge auf den Ausgang L geführt.
Die PAEs der nächsten Hierarchiestufe (1103) und auf jeder weiteren höheren Hierarchiestufe (1104, 1105) arbeiten entsprechend, weisen jedoch folgenden Unterschied auf: Der integrierte $SNDCNT_K$ ist lokal, der jeweilige Wert wird nicht weitergeleitet. $SNDCNT_K$ wird entsprechend des beschriebenen Verfahrens mit SNDCNT, dessen Wert am Eingang B anliegt synchronisiert.
SNDCNT kann zwischen sämtlichen Knoten, insbesondere aber auch zwischen den einzelnen Hierarchiestufen, beispielsweise über Register gepipelinet sein.

Zusammenfügen mittels Speicher (Memory Merge)

Bei diesem Verfahren werden zum Zusammenführen von Datenströmen Speicher verwendet. Dabei wird jedem Wert der Timestamp ein Speicherplatz zugewiesen. Die Daten werden dann entsprechend des Wertes ihrer Timestamp in dem Speicher abgelegt; mit anderen Worten dient die Timestamp als Adresse der Speicherstelle für die zugeordneten Daten. Dadurch entsteht ein Datenraum der linear zu den Timestamp ist, d.h. entsprechend

der Timestamp sortiert ist. Erst wenn der Datenraum vollständig ist, d.h. alle Daten gespeichert wurden, wird der Speicher zur weiteren Verarbeitung freigegeben, bzw. linear ausgelesen. Dies ist kann beispielsweise einfach festgestellt werden, indem gezählt wird, wieviele Daten in einen Speicher geschrieben wurden. Wurden soviele Daten geschrieben wie der Speicher Dateneinträge besitzt ist er voll.

Bei der Durchführung des Grundprinzipes tritt folgendes Problem auf: Bevor der Speicher lückenlos gefüllt ist, kann ein Überlauf der Timestamp stattfinden. Ein Überlauf ist folgendermaßen definiert: Eine Timestamp ist eine Zahl aus einem endlichen linearen Zahlenraum (TSR). Die Vergabe von Timestamps erfolgt streng monoton, wodurch innerhalb des Zahlenraumes TSR jede vergebene Timestamp eindeutig ist. Wird bei der Vergabe einer Timestamp das Ende des Zahlenraumes erreicht wird die Vergabe beim Beginn von TSR fortgesetzt; dadurch entsteht eine Unstetigkeitsstelle. Die nunmehr vergebenen Timestamp sind nicht mehr eindeutig gegenüber den vorhergehenden. Es ist grundsätzlich sicherzustellen, daß diese Unstetigkeitsstellen bei der Verarbeitung berücksichtigt werden. Der Zahlenraum (TSR) ist daher so groß zu wählen, daß im ungünstigsten Fall keine Doppeldeutigkeit dadurch entsteht, daß zwei identische Timestamp innerhalb der Datenverarbeitung auftreten. Mit anderen Worten muß TSR so groß sein, daß im ungünstigsten Fall, der innerhalb der nachfolgenden Verarbeitungspipelines und/oder Speicher auftreten kann, keine identischen Timestamp innerhalb der Verarbeitungspipelines und/oder Speicher existieren.

Tritt nunmehr ein Überlauf der Timestamp auf, müssen die Speicher in jedem Fall darauf reagieren können. Es muss davon ausgegangen werden, daß nach einem Überlauf teilweise Daten mit der Timestamp vor dem Überlauf („alte Daten") und teil-

weise Daten mit der Timestamp nach dem Überlauf („neue Daten") in den Speichern enthalten sein werden.
Die neuen Daten können nicht in die Speicherstellen der alten Daten geschrieben werden, da diese noch nicht ausgelesen wurden. Daher sind mehrere (mindestens zwei) unabhängige Speicherblöcke vorzusehen, sodaß die alten und neuen Daten getrennt geschrieben werden können.
Zur Verwaltung der Speicherblöcke können beliebige Verfahren eingesetzt werden. Zwei Möglichkeiten werden nähers ausgeführt:

a) Sofern immer sichergestellt ist, daß die alten Daten eines bestimmten Timestamp-Wertes vor den neuen Daten dieses Timestamp-Wertes eintreffen, wird getestet, ob die Speicherstelle für die alten Daten noch frei ist. Ist dies der Fall liegen alte Daten vor und die Speicherstelle wird beschrieben, wenn nicht liegen neue Daten an und die Speicherstelle für die neuen Daten wird beschrieben,

b) Ist nicht sichergestellt, daß die alten Daten eines bestimmten Timestamp-Wertes vor den neuen Daten dieses Timestamp-Wertes eintreffen, kann die Timestamp mit einer Kennung versehen werden, die alte und neue Timestamps unterscheidet. Diese Kennung kann ein oder mehrere Bits breit sein. Bei einer Überlauf der Timestamp wird die Kennung linear verändert. Dadurch sind nunmehr alte und neue Daten mit eindeutigen Timestamps versehen. Entsprechend der Kennung werden die Daten einem von mehreren Speicherblöcken zugewiesen.

Bevorzugt werden daher Kennungen verwendet, deren maximaler numerischer Werte berächtlich kleiner sind als der maximale

numerische Wert der Timestamps. Ein bevorzugtes Verhältnis kann durch folgende Formel angegeben werden:

$Kennung_{max} < TimeStamp_{max} / 2$

Verwendung von Speichern zur Partitionierung breiter Graphen

Wie aus PACT13 bekannt ist es erforderlich, große Algorithmen zu paritionieren, d.h. in eine Mehrzahl von Teilalgorithmen zu zerteilen, damit dies jeweils auf eine vorgegebene Anordnung und Menge von PAEs einer VPU passen. Die Partitionierung ist dabei einerseits performanceeffizient andererseits natürlich unter Beibehaltung der Korrektheit des ALgorithmus durchzuführen. Ein wesentlicher Aspekt ist dabei die Verwaltung der Daten und Zustände (Trigger) der jeweiligen Datenpfade. Im Folgenden werden Verfahren zur verbesserten und vereinfachten Verwaltung vorgestellt.

In vielen Fällen ist es nicht möglich einen Datenflußgraphen an nur einer Kante zu schneiden (Beispiels siehe Figur 12a), da der Graph beispielsweise zu breit ist, bzw. da zu viele Kanten (1201, 1202, 1203) an der Schnittstelle (1204) vorhanden sind.
Die Partitionierung kann erfindungsgemäß derart durchgeführt werden, daß entsprechend Figur 12b entlang aller Kanten geschnitten wird. Die Daten jeder Kante einer ersten Konfiguration (1213) werden in einen separaten Speicher (1211) geschrieben.

Es soll ausdrücklich angemerkt werden, daß zusammen mit (oder ggf. auch getrennt von) den Daten auch sämtliche relevanten Statusinformationen der Datenverarbeitung über die Kanten (beispielsweise in Figur 12b) laufen und in die Speicher geschrieben werden können. Die Statusinformationen sind in der

VPU-Technologie beispielsweise durch Trigger (vgl. PACT08) repräsentiert.

Nach der Rekonfiguration werden die Daten und/oder Statusinformationen von einer nachfolgeden Konfiguration (1214) aus den Speichern ausgelesen und durch diese Konfiguration weiterbearbeitet.

Die Speicher arbeiten als Datenempfänger der ersten Konfiguration (also in einer hauptsächlich beschreibenden Arbeitsweise) und als Datensender der nachfolgenden Konfiguration (also in einer hauptsächlich auslesenden Arbeitsweise). Die Speicher (1211) selbst sind Teil/Ressourcen beider Konfigurationen.

Zur korrekten Weiterverarbeitung der Daten ist es notwendig die ordentliche zeitliche Reihenfolge zu kennen, in der die Daten in die Speicher geschreiben wurden.
Grundlegend kann dies gewährleistet werden, indem die Datenströme entweder

a) beim Einschreiben in einen Speicher sortiert werden und/oder

b) beim Auslesen aus einem Speicher sortiert werden und/oder

c) die Sortierreihenfolge mit den Daten gespeichert und der nachfolgenden Datenverarbeitung zur Verfügung gestellt wird.

Dazu werden den Speichern Steuereinheiten zugeordnet, die sowohl beim Einschreiben der Daten (1210) in die Speicher (1211), als auch beim Auslesen der Daten aus den Speichern (1212) für die Verwaltung der Datenreihenfolgen und Datenabhängigkeiten sorgen. Je nach Ausgestaltung können unter-

schiedliche Verwaltungsarten und entsprechende Steuermechanismen eingesetzt werden.

Zwei mögliche entsprechende Verfahren sollen anhand Figur 13 ausführlicher erläutert werden. Die Speicher sind ähnlich des Datenverarbeitungsverfahren nach PACT04 einem Array (1310, 1320) aus PAEs zugeordnet:

a) In Figur 13a generieren die Speicher ihre Adressen synchron beispielsweise durch gemeinsame Adressgeneratoren und unabhängige aber synchrongeschaltete. Mit anderen Worten wird die Schreibadresse (1301) pro Zyklus weitergezählt, unabhängig ob ein Speicher tatsächlich gültige Daten zu speichern hat. Damit besitzen eine Mehrzahl von Speichern (1303, 1304) dieselbe Zeitbasis, bzw. Schreib/Leseadresse. Ein zusätzliches Flag (VOID, 1302) je Datenspeicherstelle im Speicher zeigt an, ob gültige Daten in eine Speicheradresse geschrieben wurden. Das Flag VOID kann durch das den Daten zugeordnete RDY (1305) generiert werden, entsprechend wird beim Auslesen eines Speichers das Daten RDY (1306) aus dem Flag VOID generiert. Für das Auslesen der Daten durch die nachfolgende Konfiguration wird entsprechend dem Einschreiben der Daten eine gemeinsame Leseadresse (1307) generiert, die pro Zyklus weitergeschaltet wird.

b) Effizienter ist im Beispiel nach Figur 13b ist die Zuordnung einer Timestamp zu jedem Datenwort entsprechend des bereits beschriebenen Verfahrens. Die Daten (1317) werden mit der dazugehörenden Timestamp (1311) in der jeweiligen Speicherstelle gespeichert. Dadurch entstehen keine Lücken in den Speichern und diese werden effizienter ausgelastet. Jeder Speicher weist unabhängige Schreibzeiger (1313, 1314) für die dateneinschreibende Konfiguration und Lesezeiger (1315, 1316) für die nachfolgend datenauslesende Konfiguration auf. Ent-

sprechend der bekannten Verfahren (beispielsweise entsprechend Figur 7a oder Figur 11) wird beim Auslesen das jeweils zeitlich korrekte Datenwort anhand der zugeordneten und mitgespeicherten Timestamp (1312) selektiert.

Die Sortierung der Daten in die Speicher /aus den Speichern kann also nach unterschiedlichen algorithmisch jeweils geeigneten Verfahren erfolgen, wie beispielsweise durch

a) Zuweisung eines Speicherplatzes durch die TimeStamp
b) Einsortieren in den Datenstrom gemäß der TimeStamp
c) Speichern je Takt zusammen mit einem VALID-Flag
d) Speichern der TimeStamp und derer Weitergabe an den nachfolgenden Algorithmus beim Auslesen des Speichers

Applikationsabhängig können mehrere (oder alle) Datenpfade auch vor Speichern über die erfindungsgemäßen Merge-Verfahren zusammengeführt werden. Ob dies durchgeführt wird hängt im wesentlichen von den zur Verfügung stehenden Ressourcen ab. Stehen zu wenig Speicher zur Verfügung ist eine Zusammenführung vor den Speichern erforderlich oder wünschenswert. Stehen zu wenig PAEs zur Verfügung, werden bevorzugt keine weiteren PAEs für einen Merge verwendet.

Erweiterung der Peripherieinterface (IO) durch Timestamp

Nachfolgend wird eine Verfahren beschrieben, um IO Kanälen zu Peripheriebausteinen und/oder externen Speichern Timestamps zuzuordnen. Die Zuordnung kann unterschiedliche Zwecke erfüllen, beispielsweise um eine korrekte Sortierung von Datenströmen zwischen Sender und Empfänger zuzulassen und/oder Quellen und/oder Ziele von Datenströmen eindeutig zu selektieren.

Die nachfolgenden Ausführungen werden am Beispiel der Interfacezellen aus PACT03 verdeutlicht. PACT03 beschreibt ein Verfahren zur Bündelung von VPU internen Bussen und dem Datenaustausch zwischen verschiedenen VPUs oder VPUs und Peripherie (IO).

Ein Nachteil des Verfahrens ist, daß die Datenquelle beim Empfänger nicht mehr identifizierbar und auch die korrekte zeitliche Reihenfolge nicht sichergestellt ist.
Folgende neue Verfahren lösen dieses Problem, es können jeweils anwendungsspezifisch einige oder mehrere der beschriebenen Methoden verwendet und ggf. kombiniert werden:

a) Indentifikation der Datenquelle
Figur 14 beschreibt beispielhaft eine derartige Identifikation zwischen Arrays (PAs, 1408) aus rekonfigurierbaren Elementen (PAEs) zweier VPUs. (1410, 1420) Ein Arbiter (1401) selektiert auf einem datensendenden Baustein (VPU, 1410) eine der möglichen Datenquellen (1405), um diese über einen Multiplexer (1402) auf die IO zu schalten. Die Adresse der Datenquelle (1403) wird zusammen mit den Daten (1404) an die IO gesendet. Der datenempfangende Baustein (VPU, 1411) selektiert entsprechend der Adresse (1403) der Datenquelle den entsprechenden Empfänger (1406) über einen Demultiplexer (1407). Bevorzugt kann mittels eines Übersetzungsverfahrens, z.B. einer Lookup-Tabelle, die beispielsweise durch eine übergeordnete Konfigurationseinheit (CT) konfigurierbar ist, eine flexible Zuordnung zwischen übertragener Adresse (1403) und Empfänger (1406) ermöglicht werden.
Es soll ausdrücklich darauf hingewiesen werden, dass den Multiplexern (1402) vorgeschaltet und/oder den Demultiplexern (1407) nachgeschaltet Interface-Baugruppen entsprechend

PACT03 und/oder PACT15 zur konfigurierbaren Aufschaltung von Bussystemen verwendet werden können.

b) Einhaltung der zeitlichen Reihenfolge

b1) Das einfachste Verfahren ist, die Timestamp an die IO zu senden und die Auswertung dem Empfänger zu überlassen, der die Timestamp entgegennimmt.

b2) In einer anderen Version wird die Timestamp vom Arbiter dekodiert, der nur den Sender mit der korrekten Timestamp selektiert und an die IO sendet. Der Empfänger erhält die Daten in der korrekten Reihenfolge.

Die Verfahren nach a) und b) sind entsprechend den Anforderungen der jeweiligen Applikation zusammen oder auch einzeln anwendbar.

Weiterhin kann das Verfahren durch die Vergabe und Identifikation von Kanalnummern erweitert werden. Eine Kanalnummer kennzeichnet einen bestimmten Senderbereich. Beispielsweise kann eine Kanalnummer aus mehreren Identifikationen bestehen, wie Angabe des Busses innerhalb eines Bausteines, des Bausteines, der Bausteingruppe. Dadurch ist auch in Anwendungen mit einer großen Anzahl von PAEs und/oder einem Zusammenschluß vieler Bausteine eine einfache Identifizierung gegeben.

Vorzugsweise werden bei der Verwendung von Kanalnummern nicht jeweils einzelne Datenworte übertragen, sondern eine Mehrzahl von Datenworten werden zu einem Datenpaket zusammengefaßt und sodann unter Angabe der Kanalnummer übertragen. Das Zusammenfassen der einzelnen Datenworte kann beispielsweise mittels

eines geeigneten Speichers erfolgen, wie z.B. in PACT18 (BURST-FIFO) beschrieben.

Es soll darauf hingewiesen werden, dass die übertragenen Adressen und/oder Timestamp bevorzugt als Kennungen oder Teil von Kennungen in Bussystemen nach PACT15 eingesetzt werden können.

Vollumfänglich wird das Verfahren nach PACT07 in das Patent eingegliedert, welches durch das beschriebene Identifizierungsverfahren erweitert werden kann. Weiterhin sind die Datenübertragungsverfahren nach PACT18 vollumfänglich eingegliedert, für die das beschriebene Verfahren ebenfalls Anwendung finden kann.

Aufbau von Sequenzern

Durch den Einsatz von TimeStamps oder vergleichbaren Verfahren wird ein einfacher Aufbau von Sequenzern aus Gruppen von PAEs ermöglicht. Die Busse und Grundfunktionen der Schaltung werden konfiguriert, die Detailfunktion und Datenadressen werden durch einen OpCode zur Laufzeit flexibel eingestellt.

Mehrere diese Sequenzer lassen sich zugleich innerhalb eines PA (Arrays aus PAEs) aufbauen und betreiben.

Die Sequenzer innerhalb einer VPU sind entsprechend des Algorithmus aufbaubar, Beispiele wurden in mehreren vollumfänglich eingegliederten Schriften des Erfinders bereits gegeben. Insbesondere soll auf PACT13 verwiesen werden, worin der Aufbau von Sequenzern aus einer mehrzahl von PAEs beschrieben ist, was als beispielhafte Grundlage für die nachfolgende Beschreibung dient.

Im Detail lassen sich zum Beispiel folgenden Ausgestaltungen von Sequenzern frei anpassen:

- Art und Menge der IO/Speicher
- Art und Menge der Interrupts (z.B. über Trigger)
- Befehlssatz
- Anzahl und Art von Registern

Ein einfacher Sequenzer läßt sich beispielsweise aus

1. einer ALU zur Durchführung der arithmetischen und logischen Funktionen
2. einem Speicher zur Speicherung der Daten, quasi als Registersatz
3. einem Speicher als CodeQuelle für das Programm (z.B. normaler Speicher nach PACT22/24/13 und/oder CT nach PACT10/PACT13 und/oder speziellen Sequenzern nach PACT04)

aufbauen

Ggf. wird der Sequenzer um IO-Elemente (PACT03, PACT22/24) erweitert. Zudem lassen sich weitere PAEs als Datenquellen oder -empfänger anschließen.

Je nach eingesetzter CodeQuelle kann das Verfahren nach PACT08 zum Einsatz kommen, das das direkte Setzen von OpCodes einer PAE über Datenbusse, sowie die Angabe der Datenquellen/-ziele zuläßt.

Die Adressen der Datenquellen/-ziele lassen sich beispielsweise per TimeStampverfahren übertragen. Weiterhin kann der Bus zur Übertragung der OpCodes verwendet werden.

In einer beispielhaften Implementierung nach Figur 15 besteht ein Sequenzer aus einem RAM zur Speicherung des Programmes

(1501), einer PAE zur Berechnung der Daten (ALU) (1502), einer PAE zur Berechnung des Programmzeigers (1503), einem Speicher als Registersatz (1504) und einer IO für externe Geräte (1505).

Durch Verdrahtung entstehen zwei Bussysteme, ein Eingangsbus zur ALU IBUS (1506) und ein Ausgangsbus von der ALU OBUS (1507). Den Bussen zugeordnet ist jeweils eine 4-bit breite Timestamp, die die Quelle IBUS-ADR (1508) bzw. das Ziel OBUS-ADR (1509) adressiert.

Von 1504 wird der Programmzeiger (1510) an 1501 geführt. 1501 liefert den OpCode (1511) zurück. Der OpCode wird in Befehle für die ALU (1512) und den Programmzeiger (1513), sowie die Datenadressen (1508, 1509) aufgespaltet. Zur Aufspaltung des Busses können die nachfolgend beschriebenen SIMD Verfahren und Bussysteme verwendet werden.

1502 ist als Akkumulatormaschine ausgestaltet und unterstützt beispielsweise folgende Funktionen:

ld <reg>	Lade Akkumulator (1520) von Register
add_sub <reg>	Addiere/Subtrahiere Register zum Akkumulator
sl_sr	Akkumulator schieben
rl_rr	Akkumulator rotieren
st <reg>	Schreibe Akkumulator in Register

Für die Befehle sind 3-bit notwendig. Ein viertes Bit gibt die Art der Operation an: addieren oder subtrahieren, rechts- oder linksschieben.

1502 liefert den ALU-Status carry auf Triggerport 0 und zero auf Triggerport 1.

<reg> ist folgendermaßen kodiert:

0..7	Datenregister in 1504
8	Eingangsregister (1521) Programmzeigerberechnung

9 IO-Daten

10 IO-Adressen

Für die Adressen werden 4 bits benötigt.

1503 unterstützt folgende Operationen über den Programmzeiger:

jmp	Springe auf Adresse im Eingangsregister (2321)
jt0	Springe auf Adresse im Eingangsregister angegeben ist, wenn Trigger0 gesetzt
jt1	Springe auf Adresse im Eingangsregister angegeben ist, wenn Trigger1 gesetzt
jt2	Springe auf Adresse im Eingangsregister angegeben ist, wenn Trigger2 gesetzt
jmpr	Springe zu PP plus Adresse im Eingangsregister

Für die Befehle sind 3-bit notwendig. Ein viertes Bit gibt die Art der Operation an: addieren oder subtrahieren.

Der OpCode 1511 wird also in 3 Gruppen zu je 4 bit zerlegt: (1508, 1509), 1512, 1513, 1508 und 1509 können bei dem gegebenen Instruktionssatz identisch sein. 1512, 1513 werden beispielsweise an das C-Register der PAEs geführt (vgl. PACT22/24) und innerhalb der PAEs als Befehl dekodiert (vgl. PACT08).

Entsprechend PACT13 und/oder PACT11 kann der Sequenzer in eine komplexere Struktur eingebaut werden. Beispielsweise sind durch <reg> = 11, 12, 13, 14, 15 weitere Datenquellen adressierbar, die auch von anderen PAEs stammen können. Ebenfalls lassen sich weitere Datenempfänger adressieren. Datenquellen und -empfänger können beliebig, insbesondere PAEs sein.

Es soll angemerkt sein, daß die aufgezeigte Schaltung nur 12-bit des OpCodes 1511 benötigt. Bei einer 32-bit Architektur

stehen damit 20-bit optional zur Erweiterung der Grundschaltung zur Verfügung.

Die Multiplexerfunktionen der Busse können entsprechend des beschriebenen TimeStampverfahrens implementiert werden. Andere Ausgestaltungen sind ebenfalls möglich, beispielswiese könnten PAEs als Multiplexerstufen eingesetzt werden.

SIMD Rechenwerke und SIMD Bussysteme

Bei der Verwendung von rekonfigurierbaren Technologien zur Verarbeitung von Algorithmen entsteht ein wesentliches Paradoxon: Einserseits sind um eine möglichst hohe Rechenleistung zu erhalten komplexe ALUs erforderlich, wobei der Aufwand für die Rekonfiguration minimal sein sollte; andererseits sollten die ALUs möglichst einfach sein um effiziente Verarbeitung auf Bitebene zu ermöglichen; andererseits sollte die Rekonfiguration und Datenverwaltung derart intelligent und schnell erfolgen, daß sie effizient und einfach zu programmieren ist.

Bisherige Technologien verwenden a) sehr kleine ALUs mit wenig Rekonfigurationsunterstützung(FPGAs) und sind auf Bitebene effizient, b) große ALUs (Cameleon) mit wenig Rekonfigurationsunterstützung, c) eine Mischung aus großen ALUs und kleinen ALUs mit Rekonfigurationsunterstützung und Datenverwaltung (VPUs).

Da die VPU-Technologie die leistungsfähigste Technik darstellt, soll auf ihr aufbauend ein optimiertes Verfahren geschaffen werden. Es soll ausdrücklich darauf hingewiesen werden, daß dieses Verfahren ebenfalls für die anderen Architekturen eingesetzt werden kann.

Der Flächenaufwand zur effizienten Steuerung von Rekonfiguration ist mit einer Menge von ca. 10.000 bis 40.000 Gattern pro PAE vergleichsweise hoch. Unterhalb dieser Gattermenge lassen sich nur einfache Ablaufsteuerungen realsieren, die die Programmierbarkeit von VPUs erheblich einschränken und eine Verwendung als General Purpose Prozessor ausschließen. Sofern auf eine besonders schnelle Rekonfiguration abgezielt wird, müssen zusätzliche Speicher vorgesehen werden, wodurch die erforderliche Gattermenge nochmals erheblich ansteigt.

Um ein ordentliches Verhältnis zwischen Rekonfigurationsaufwand und Rechenleistung zu erhalten, ist somit der Einsatz von großen ALUs (viel Funktionalität und/oder große Bitbreite) zwingend erforderlich. Werden die ALUs jedoch zu groß, sinkt die nutzbare parallele Rechenleistung pro Chip. Bei zu kleinen ALUs (z.B. 4-bit) ist der Aufwand zur konfiguration aufwendiger Funktionen (z.B. 32-bit Multiplikation) zu hoch. Insbesondere der Verdrahtungsaufwand wächst in kommerziell nicht mehr sinnvolle Bereiche.

11.1 Einsatz von SIMD Rechenwerken

Um ein ideales Verhältnis zwischen der Verarbeitung von kleinen Bitbreiten, Verdrahtungsaufwand und der Konfiguration aufwendiger Funktionen zu erhalten, wird der Einsatz von SIMD-Rechenwerken vorgeschlagen. Dabei werden Rechenwerke der Breite m derart zerteilt, daß n einzelne Blöcke der Breite b = m/n entstehen. Durch Konfiguration wird je Rechenwerk vorgegeben, ob ein Rechenwerk unzerteilt arbeiten soll, oder ob das Rechenwerk in einen oder mehrere Blöcke, jeweils gleicher oder unterschiedlicher Breite zerlegt sein soll. Mit anderen Worten, kann ein Rechenwerk auch derart zerlegt sein, daß innerhalb eines Rechenwerkes unterschiedliche Wortbreiten zu-

gleich konfiguriert sind (z.B. Breite 32-bit, zerlegt in 1x16-, 1x8- und 2x4-bit). Die Daten werden derart zwischen den PAEs übertragen, daß die zerlegten Datenworte (SIMD-WORD) zu Datenworten der Bitbreite m zusammengefaßt werden und als Paket über das Netzwerk übertragen werden.
Das Netzwerk überträgt immer ein komplettes Paket, d.h. alle Datenworte sind innerhalb eines Paketes gültig, und werden nach dem bekannte Handshake-Verfahren übertragen.

11.1.1 Umsortieren der SIMD-WORD

Für einen effizienten Einsatz der von SIMD-Rechenwerken ist eine flexible und effiziente Umsortierung der SIMD-WORD untereinander innerhalb eines Busses oder zwischen unterschiedlichen Bussen erforderlich.
Die Busschalter nach Figur 5 bzw. 7b,c können derart modifiziert werden, daß eine flexible Vernetzung der einzelnen SIMD-WORD möglich ist. Dazu werden die Multiplexer entsprechend den Rechenwerken teilbar ausgelegt, derart daß durch Konfiguration die Teilung bestimmt werden kann. Mit anderen Worten, wird beispielsweise nicht ein Multiplexer der Breite m pro Bus verwendet, sondern n einzelne Multiplexer der Breite b = m/n. Es ist nunmehr möglich, die Datenbusse für b bit Breite zu konfigurieren. Durch die Matrixstruktur der Busse (Figur 5) wird eine einfache Umsortierung der Daten ermöglicht, wie in Figur 16c dargestellt. Eine erste PAE sendet Daten über zwei Busse (1601, 1602) die in je 4 Teilbusse unterteilt sind. Ein Bussystem (1603) verschaltet die einzelnen Teilbusse mit zusätzlichen auf dem Bus befindlichen Teilbusse. Eine zweite PAE erhält unterschiedlich sortierte Teilbusse an ihren beiden Eingangsbussen (1604, 1605).

Die Handshakes der Busse zwischen zwei PAEs mit beispielsweise 2-fach SIMD Rechenwerk (1614, 1615) werden in Figur 16a logisch derart verknüpft, daß ein gemeinsamer Handshake (1610) für den neu geordneten Bus (1611) aus den Handshakes der ursprünglichen Busse generiert wird. Beispielsweise kann ein RDY für einen neu sortierten Bus aus einer logischen UND-Verknüpfung aller RDYs der Daten für diesen Bus liefernden Busse generiert werden. Ebenso kann beispielsweise das ACK eines Daten liefernden Busses aus einer UND-Verknüpfung der ACKs aller Busse generiert werden, die die Daten weiterverarbeiten.
Der gemeinsame Handshake steuert eine Steuereinheit (1613) für die Verwaltung der PAE (1612) an. Der Bus 1611 wird PAE intern auf zwei Rechenwerke (1614, 1615) aufgeteilt.

In einer ersten Ausführungsvariante finden die Verknüpfungen der Handshakes innerhalb eines jeden Busknotens statt. Dadurch wird es möglich einem Bussystem der Breite m, bestehend aus n Teilbussen der Breite b, nur ein Handshake-Protokol zuzuordnen.

In einer weiteren besonders bevorzugten Ausgestaltung werden sämtliche Bussysteme in der Breite b ausgestaltet, die die kleinste realisierbare Ein-/Ausgabe Datenbreite b eines SIMD-Word entspricht. Entsprechend der Breite der PAE Datenpfade (m) besteht nunmehr ein Ein-/Ausgangsbus aus m/b = n Teilbussen der Breite b. Beispielsweise besitzt eine PAE mit 3 32bit Eingangsbussen und 2 32 bit Ausgangsbusse bei einer kleinsten SIMD-Wortbreite von 8 tatsächlich 3x4 8bit Eingangsbusse und 2x4 8bit Ausgangsbusse.
Jedem der Teilbusse sind sämtliche Handshake- und Steuersignale zugeordnet.

Der Ausgang einer PAE versendet die mit denselben Steuersignalen für sämtliche n Teil-Busse. Eingehende Quittierungssignale aller Teilbusse werden miteinander logische verknüpft, z.B. durch eine UND-Funktion. Die Bussysteme können jeden Teilbus frei verschalten und unabhängig routen. Die Bussystem und insbesondere die Busknoten verarbeiten und verknüpfen die Handshake-Signale der einzelnen Busse unabhängig ihres Routings, ihrer Anordung und Sortierung nicht.
Bei einer PAE eingehenden Daten werden die Steuersignale sämtlicher n Teilbusse derart miteinander verknüpft, dass ein allgemeingültiges Steuersignal quasi als Busssteuersignal für den Datenpfad generiert wird.
Beispielsweise können in einer "dependend" Betriebsart gemäß Definition RdyHold-Stufen für jeden einzelnen Datenpfad eingesetzt werden und erst wenn sämtliche RdyHold-Stufen anstehende Daten signalisieren, werden diese von der PAE übernommen.
In einer "independend" Betriebsart gemäß Definition werden die Daten jedes Teilbusses einzeln in Eingangsregister der PAE geschrieben und quittiert, wodurch der Teilbus sofort für eine nächste Datenübertragung frei ist. Das Vorhandensein aller erforderlichen Daten von allen Teilbussen in den Eingangsregistern wird innerhalb der PAE durch geeignete logische Verknüpfung der für jeden Teilbus im Eingangsregister gespeicherten RDY-Signale detektiert, woraufhin die PAE mit der Datenverarbeitung beginnt.

Der wesentliche daraus resultierende Vorteil dieses Verfahrens ist, dass die SIMD-Eigenschaft von PAEs keinerlei besonderen Einfluß auf das verwendete Bussystem aufweist. Es werden lediglich wie in Figur 16b dargestellt mehr Busse (n) (1620) einer geringeren Breite (b) und die zugeordneten Handshakes (1621) benötigt. Die Verschaltung selbst bleibt

unberührt. Die PAEs verknüpfen und verwalten die Steuerleitungen lokal. Dadurch entfällt der zusätzliche Hardwareaufwand in den Bussystemen zur Verwaltung und/oder Verknüpfung der Steuerleitungen.

Titel: Verfahren zur Steuerung der Übertragung von Datenströmen

Patentansprüche

1. Verfahren zum Steuern von pipelineartigen Datenverarbeitungen und/oder Bussystemen dadurch gekennzeichnet, dass abwechselnd unterschiedliche Protokolle angewendet werden, um eine Datenverarbeitung in jedem Takt zu ermöglichen.

2. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass ein Protokoll die Annahme der Daten durch einen Empfänger bestätigt.

3. Verfahren nach Anspruch 1-2, dadurch gekennzeichnet, dass ein Protokoll die voraussichtliche Annahme der Daten durch einen Empfänger bestätigt.

4. Verfahren nach Anspruch 1-3, dadurch gekennzeichnet, dass wenn die für eine voraussichtliche Abnahme bestätigten Daten durch einen Empfänger nicht angenommen werden können, diese

in ein Pufferregister geschrieben werden und danach bis zur Entleerung des Pufferregisters keine weitere voraussichtliche Datenannahme durch einen Empfänger mehr bestätigt wird.

5. Verfahren nach Anspruch 1-4, dadurch gekennzeichnet, dass das Pufferregister sobald der Empfänger wieder Daten annimmt geleert wird, bevor weitere andere Daten an den Empfänger gesendet werden.

6. Verfahren zum Übertragen von Daten eines Senders an mehrere Sender, dadurch gekennzeichnet, dass
die Quittierung der Datenabnahme aller Sender logisch verknüpft wird.

7. Verfahren zum Übertragen von Daten mehrerer Senders an einen Sender, dadurch gekennzeichnet, dass
die Reihenfolge der Übertragungsanforderungen mehrerer Sender gespeichert wird und die Übertragung der Daten in der exakten Reihenfolge ermöglicht wird.

8. Verfahren zum Übertragen von Daten mehrerer Senders an einen Sender, dadurch gekennzeichnet, dass
jedem Sender bei einer erfolgten Buszugriffsanforderung eine Sendernummer zugeteilt wird, die seine Position in der Reihe der Sender bezeichnet.

9. Verfahren nach Anspruch 8, dadurch gekennzeichnet, dass der Reihe nach alle Sendernummern von einem Aufrufnummerngenerator aufgerufen werden, indem allen Sendern die aktuelle Aufrufsnummer mitgeteilt wird, jeder Sender diese mit seiner Sendenummer vergleicht und bei Übereinstimmung den Bus beansprucht.

10. Verfahren nach Anspruch 8, dadurch gekennzeichnet, dass die Sendenummern pro Zeiteinheit weitergezählt werden.

11. Verfahren nach Anspruch 8-9, dadurch gekennzeichnet, dass der Bus arbitriert wird, wenn mehrere Sender dieselbe Sendernummer zugeteilt haben.

12. Verfahren nach Anspruch 8-9 und 11, dadurch gekennzeichnet, dass
der Aufrufnummerngenerator erst weiterzählt wenn kein Sender den Bus mehr arbitriert hat.

13. Verfahren zum Verwalten von Datenströmen, dadurch gekennzeichnet, dass
den Daten eine Kennung zugewiesen wird.

14. Verfahren nach Anspruch 13, dadurch gekennzeichnet, dass die Kennung eine zeitliche Reihenfolge festlegt.

15. Verfahren nach Anspruch 13, dadurch gekennzeichnet, dass die Kennung eine Herkunfts- oder Zieladresse festlegt.

16. Verfahren nach Anspruch 13-14, dadurch gekennzeichnet, dass
anhand der Kennung das Zusammenführen der Daten in der ursprünglichen Reihenfolge mittels eines Bussystems definiert wird.

17. Verfahren nach Anspruch 13-14, dadurch gekennzeichnet, dass
anhand der Kennung das Zusammenführen der Daten in der ursprünglichen Reihenfolge mittels eines Speichers definiert wird.

18. Verfahren nach Anspruch 13-17, dadurch gekennzeichnet, dass
die Kennung über die peripheren Interface versendet wird.

19. Verfahren nach Anspruch 13-17, dadurch gekennzeichnet, dass
die Kennung in Speicher zusammen mit den Daten geschrieben wird.

20. Verfahren zum Partitioniern von Graphen, dadurch gekennzeichnet, dass
an den Schnittkanten Speicher eingefügt werden.

21. Verfahren nach Anspruch 20, dadurch gekennzeichnet, dass
je Kante eines Graphen ein Speicher verwendet wird.

22. Verfahren nach Anspruch 20, dadurch gekennzeichnet, dass
Multiplexer mehrere Kanten vor einem Speicher zusammenfassen.

23. Verfahren nach Anspruch 20-22, dadurch gekennzeichnet, dass
eine Kennung mit den Daten gespeichert wird.

24. Verfahren zum Aufbau von Sequenzern aus mehreren PAEs, dadurch gekennzeichnet, dass
die den Daten zugeordnete Kennung zur Adressierung von Datenquellen und/oder Datenzielen dient.

25. Verfahren zum Aufbau von Sequenzern aus mehreren PAEs, dadurch gekennzeichnet, dass
die den Daten zugeordnete Kennung den Datenverarbeitungbefehl beinhaltet.

26. Pipelineartiges Datenverarbeitungsverfahren, dadurch gekennzeichnet, dass
den Datenverarbeitungselementen FIFO-Puffer zur zeitlichen Entkopplung zwischengeschaltet sind.

27. Verfahren nach Anspruch 26, dadurch gekennzeichnet, dass die FIFO-Puffer konfigurierbare Latenzzeiten aufweisen, um Datenpfade in ihrer Verzögerung auszubalancieren.

28. FIFO-Speicherverfahren, dadurch gekennzeichnet, dass der Auslesevorgang bei einem früher ausgelesenen Datenwort wieder aufgenommen werden kann.

29. FIFO-Speicherverfahren, dadurch gekennzeichnet, dass der Einschreibvorgang bei einem früher eingeschriebenen Datenwort wieder aufgenommen werden kann.

30. Verfahren nach Anspruch 28-29, dadurch gekennzeichnet, dass
ein Sicherungsregister die Adressposition des Datenwortes sichert, an dessen Adresse ein Vorgang wiederholt werden kann.

31. Verfahren nach Anspruch 28-30, dadurch gekennzeichnet, dass
der Leer- oder Vollzustand des FIFOs durch Vergleich mit dem Sicherungsregister getestet wird.

32. Verfahren nach Anspruch 28-31, dadurch gekennzeichnet, dass
das Sicherungregister beliebig auf jede Adresse gesetzt werden kann.

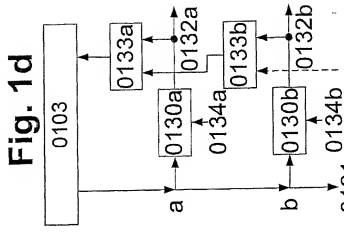

Fig. 1d

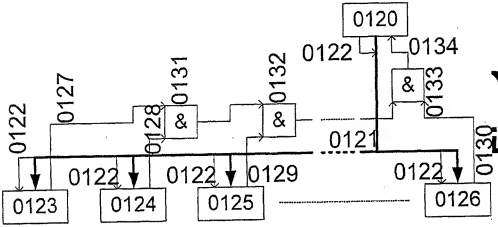

Fig. 1c

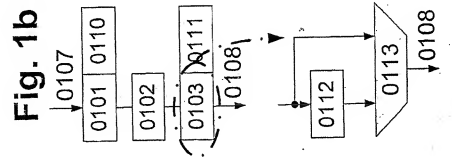

Fig. 1b

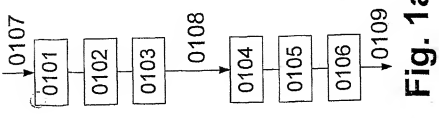

Fig. 1a

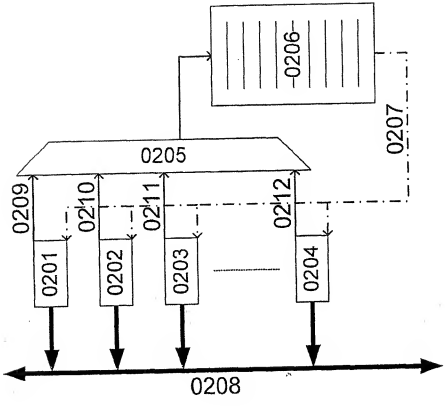

Fig. 2

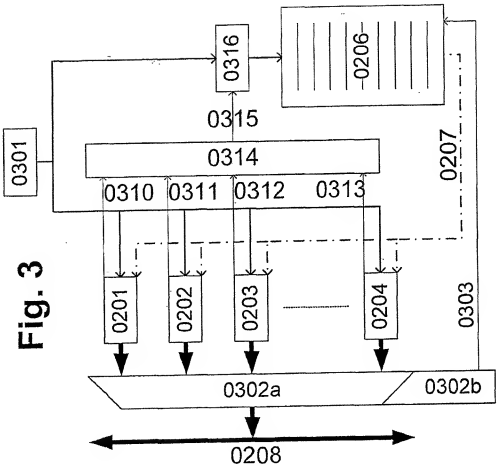
0316
0206
0315
0301
0314
0207
0310
0311
0312
0313
0201
0202
0203
0204
0303
0302a
0302b
0208

Fig. 3

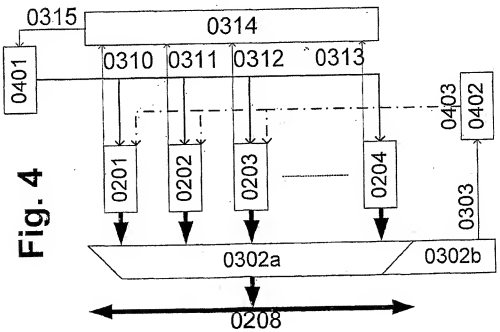
0315
0314
0401
0310
0311
0312
0313
0403
0402
0201
0202
0203
0204
0303
0302a
0302b
0208

Fig. 4

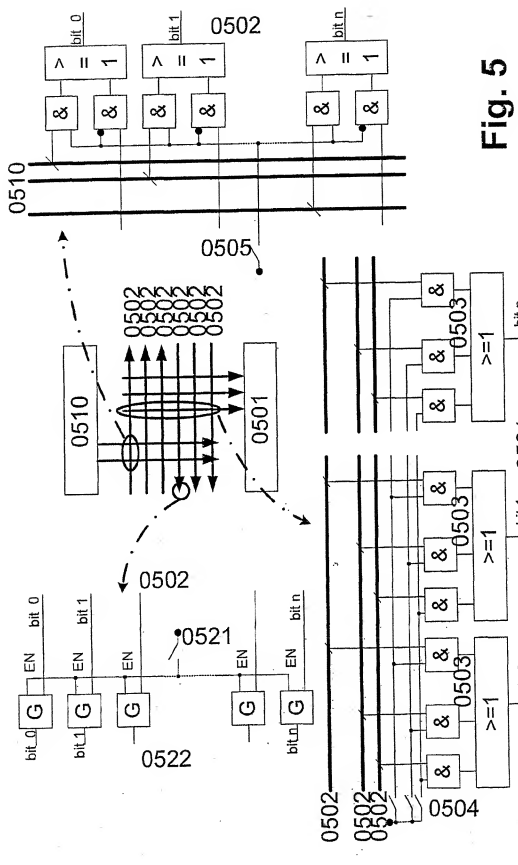

Fig. 5

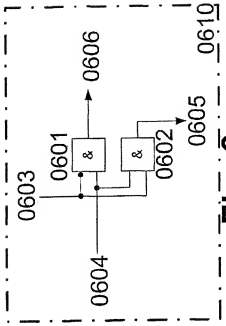
0603
0601
&
0606
&
0602
0604
0605
0610

Fig. 6a

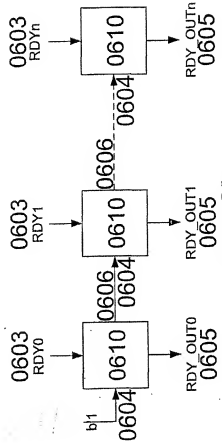
0603
RDY0
0610
0606
0604
RDY_OUT0
0605
b1
0604
0603
RDY1
0610
0606
0604
RDY_OUT1
0605
0603
RDYn
0610
0604
RDY_OUTn
0605

Fig. 6b

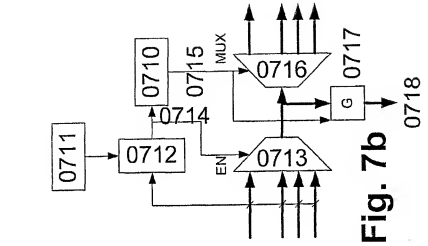
0711
0712
0714
0710
0715
MUX
0716
0717
G
0718
EN
0713

Fig. 7b

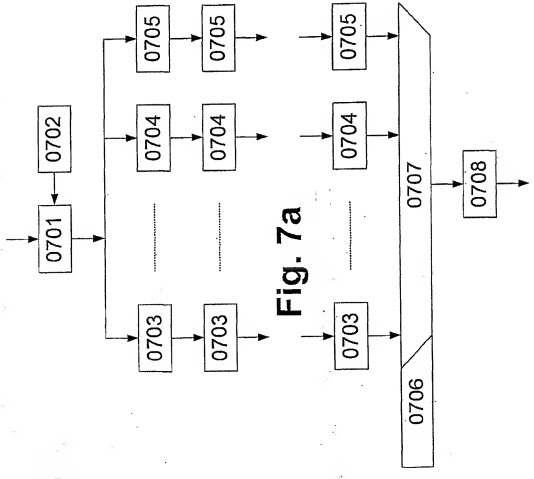
0701
0702
0703
0704
0705
0706
0707
0708

Fig. 7a

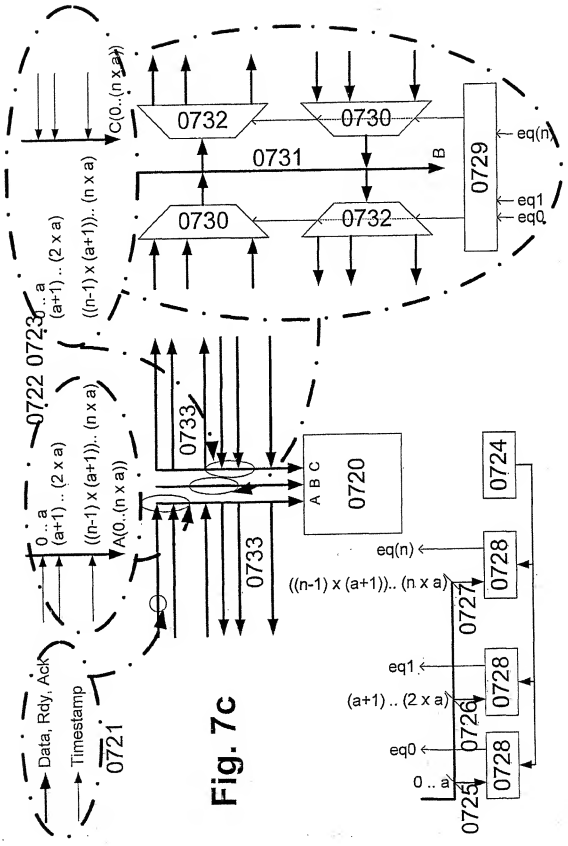
Data, Rdy, Ack
Timestamp
0721
0722
0723
0 .. a
(a+1) .. (2 x a)
((n-1) x (a+1)).. (n x a)
A(0..(n x a))
C(0..(n x a))
0733
0720
A B C
0724
0728
0727
0726
0725
eq(n)
eq1
eq0
0732
0730
0731
B
0729
eq1
eq0

Fig. 7c

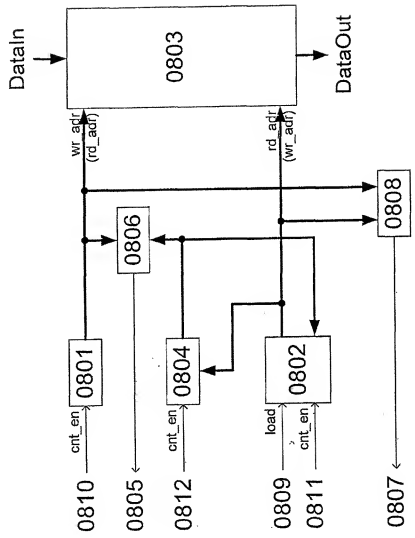
DataIn
0803
DataOut
wr_adr
(rd_adr)
rd_adr
(wr_adr)
0801
0806
0804
0802
0808
cnt_en
cnt_en
load
cnt_en
0810
0805
0812
0809
0811
0807

Fig. 8

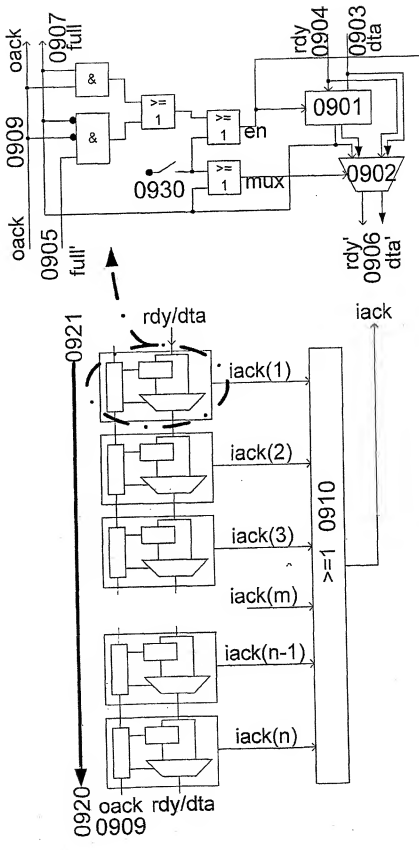
oack
0907
full
0909
oack
0905
full'
&
&
>= 1
>= 1
>= 1
en
mux
0930
0901
0902
rdy
0904
0903
dta
iack(m)
0908
rdy'
0906
dta'
0921
rdy/dta
iack(1)
iack(2)
iack(3)
iack(m)
iack(n-1)
iack(n)
iack
>=1
0910
0920
oack rdy/dta
0909

Fig. 9

Fig. 9a

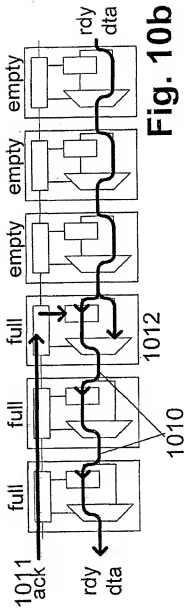

Fig. 10b

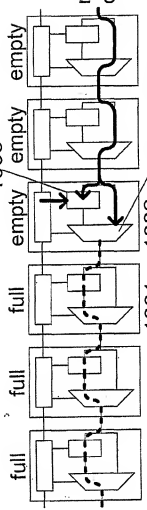

Fig. 10a

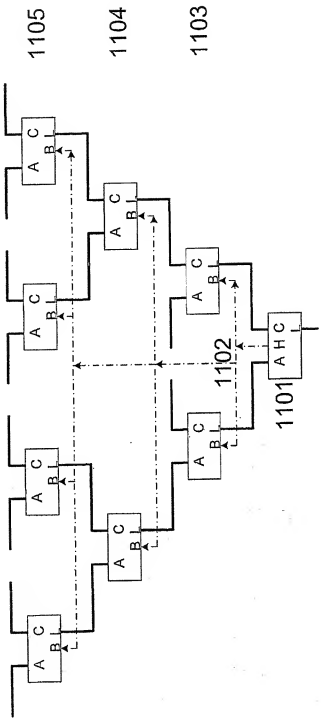

Fig. 11

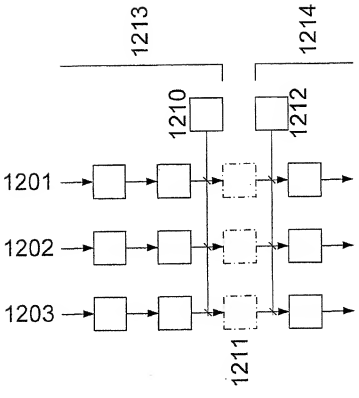

Fig. 12b

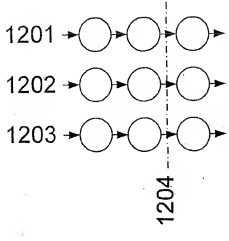

Fig. 12a

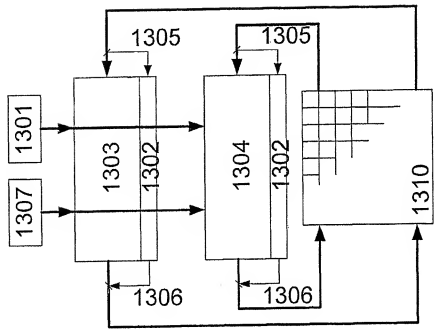

Fig. 13a

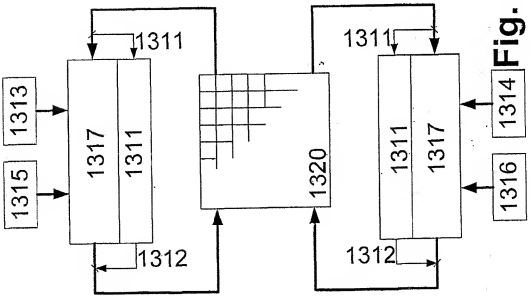

Fig. 13b

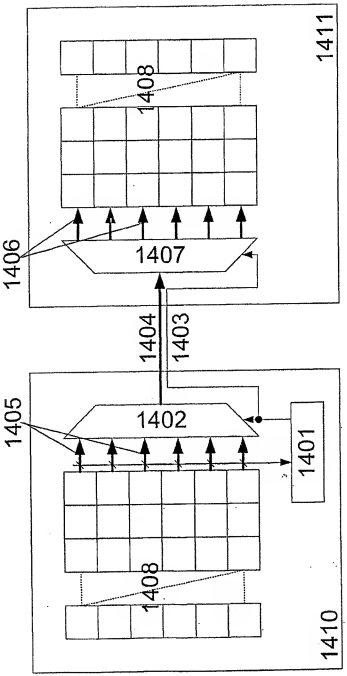

Fig. 12

MISSING AT THE TIME OF PUBLICATION

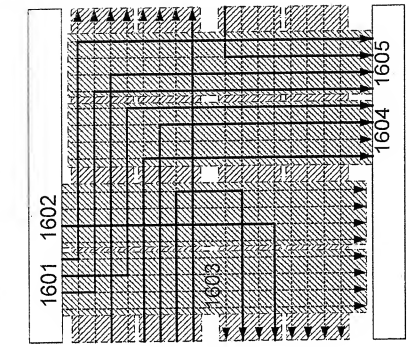

Fig. 16c

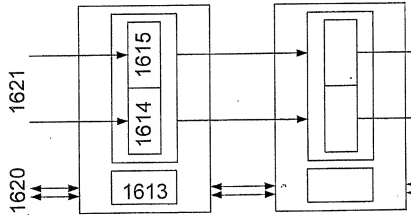

Fig. 16b

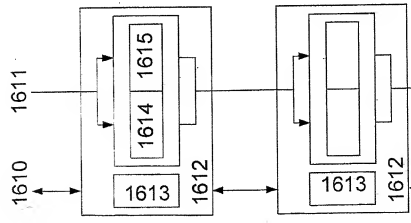

Fig. 16a